

CRANFIELD UNIVERSITY

MOSES ASHAWA APROFIN

THE DETECTION AND PREVENTION OF MALWARE ATTACKS ON
ANDROID MOBILE THROUGH THE APPLICATION OF ARTIFICIAL
INTELLIGENCE TECHNIQUES

SCHOOL OF DEFENCE AND SECURITY
PhD in DEFENCE AND SECURITY

PhD
Academic Year: 2018 - 2021

Supervisor: Dr SARAH MORRIS
Associate Supervisor: Dr VENKAT SASTRY
September 2021

CRANFIELD UNIVERSITY

SCHOOL OF DEFENCE AND SECURITY
PhD in DEFENCE AND SECURITY

PhD

Academic Year 2018 – 2021

MOSES ASHAWA APROFIN

THE DETECTION AND PREVENTION OF MALWARE ATTACKS ON
ANDROID MOBILE THROUGH THE APPLICATION OF ARTIFICIAL
INTELLIGENCE TECHNIQUES

Supervisor: Dr Sarah Morris
Associate Supervisor: Dr Venkat Sastry
September 2021

This thesis is submitted in partial fulfilment of the requirements for the degree of
Doctor of Philosophy (PhD)

© Cranfield University 2021. All rights reserved. No part of this publication may
be reproduced without the written permission of the copyright owner.

ABSTRACT

Our everyday lives are integrated with the use of mobile devices which store sensitive data. Sensitive data stored on smartphones attract different threats including malware. Among mobile platforms, Android is the most popular OS with malware targeting sensitive information and other mobile services. If malware infects a digital device, then it has control over the device's functionality and data. This can impact your finances, your privacy, and your access to your data. Malware is a threat not only to individuals but also to corporate organisations and financial institutions as well. This could lead to communication traffic of an infected network, hardware failure of the physical device, data theft, and loss of critical business data, among others. There are existing detection techniques for identifying Android malware. However, these techniques are limited in detecting evolving and sophisticated malware which use permission features as attack vectors in a smart fashion to infect Android mobile devices.

To improve malware detection accuracy based on the related problem, we developed techniques for identifying Android-based malicious applications. To achieve this, the author presents a thorough review of the mobile malware evolution and infection strategies. The second part of the survey covers Android mobile malware detection, classification, and analysis techniques where the author identifies their efficacy in detecting evolving malware and their limitations. The author identifies through the review research gaps which open unto the development of different and novel solutions for Android malware classification and analysis.

We leveraged the existing strengths of the previous methods to develop a robust novel automated framework to classify and analyse Android malware based on permission features. Classification accuracy of 97% was achieved with our framework with a False Positive Rate of 3%. Our techniques identified privileges that malware exploits as attack vectors to infect Android-based devices. The results demonstrate that our framework has high feature diversity capabilities for Android malware classification. We identified that there are permissions with similar attributes that are correlated and can trigger the installation of similar permissions with the same threat level especially. However, these prevention techniques are not tested on other mobile platforms' data and do not focus on mitigating pileup susceptibilities. Finally, we believe that as the results of this research are being made public and cited by organizations and individuals, the outcome of this will influence the security and social policies that mobile companies will implement based on some of the recommendations by our findings.

Keywords:

Android malware; Permissions; Feature diversity; Privilege exploitation; Threat level; Protection level; Attack vectors, Digital Investigation.

DEDICATION

I dedicate this thesis to God Almighty for the grace to start and finish this PhD.

ACKNOWLEDGEMENTS

I would like to express my utmost gratitude to my supervisors, Dr. Sarah Morris and Dr. Venkat Sastry for their sincere and selfless support, prompt and useful advice, feedbacks, patience, and encouragement during this research. These have kept me going against challenges. I also express my profound gratitude to my review team Dr. Gareth Appleby-Thomas and Dr. Nikki Williams for their tremendous review guidance and feedbacks. My gratitude will not be complete without Melissa Hadgkiss who provided network access to the lab and all the resources needed with the experiments. Before the viva, Melissa with my supervisor created mock viva sessions for drilling to enable me to feel comfortable on how to answer viva questions in a concise but concrete manner during the main viva. Thank you all.

It would have not been possible to carry out this research without funding. Thanks to the Petroleum Technology Development Fund (PTDF) of Nigeria for sponsoring my PhD. My special thanks go to the Cranfield University Finance Department for providing necessarily financial assistance at different points whenever they were required. My gratitude goes to the registry office and the student academic support unit of the University for supporting me at different points during my time as a PhD student. My special thanks to the Digital Investigation Unit (DIU) staff and the entire staff of the Centre for Electronic warfare, Information and Cyber. I thank the Doctoral Community especially Bea Kingdon and Dr Clare Knock for the numerous and uncountable support. My gratitude goes to the Barring library team, the Cranfield Student Association, the IT department, and all individuals that assisted at various times.

A big thank you to my parents, siblings, and friends for their moral supports and encouragement. Thank you, Dad and Mum, for your words of encouragement always saying “Moses, you have what it takes to make it happen”.

Finally, an immense thank you to fellow researchers and friends that have made my experience at Cranfield University memorable and helping along the way. I express my warm gratitude to Dr. Anthony Nyombi and Dr. Henry Agu with whom I had guidance on data presentation, analysis, discussion, and also offering advice on thesis by paper presentation and format. A big thank you to Godwin Ibrahim who assisted me when I was very ill during my PhD.

TABLE OF CONTENTS

ABSTRACT	4
DEDICATION	6
ACKNOWLEDGEMENTS	7
TABLE OF CONTENTS	8
LIST OF FIGURES.....	12
LIST OF TABLES.....	13
LIST OF EQUATIONS	14
CHAPTER 1: INTRODUCTION	20
1.1 MALWARE ATTACKS AND ECONOMIC IMPACTS	20
1.1.2 Recent cases of malware attacks	21
1.2 MOBILE DEVICES	24
1.3 ANDROID OPERATING SYSTEM	26
1.3.1 Structure of APK file	27
1.3.2 Taxonomy of APK structure	29
1.4 DIGITAL INVESTIGATION AND MOBILE MALWARE.....	30
1.5 MOTIVATION	31
1.6 RESEARCH QUESTION	34
1.7 RESEARCH OBJECTIVES.....	36
1.8 RESEARCH CONTRIBUTION TO KNOWLEDGE AND THESIS ORGANISATION	39
1.8.1 Background and techniques	39
1.8.2 Identification of attack vectors and analysis of malicious features and families	40
1.8.3 Permission variables anomaly detection	41
1.8.4 Bayesian model for identifying malware based on permission correlation.....	41
1.9 ETHICAL CONSIDERATIONS.....	42
1.9.1 We acknowledged that malware is dangerous	42
1.9.2 Obtaining malware to create a corpus.....	42
1.9.3 Emulated and physical environments.....	43
1.9.4 Dataset storage, access, and privacy	43
1.9.5 Legal issues.....	44
REFERENCES	45
CHAPTER 2: ANDROID MOBILE MALWARE EVOLUTION AND INFECTION STRATEGIES: A SURVEY	49
2.1 INTRODUCTION	50
2.2 METHODOLOGY.....	53
2.3 BACKGROUND	57
2.3.1 History of mobile malware	57
2.3.2 Evolution of Android malware	62
2.3.2.1 Dark web advancements and new malware variants	66
2.3.2.2 Smart malware evasion trends using technological techniques.....	67
2.4 CLASSIFICATION OF MOBILE MALWARE	74
2.4.1 Ransomware.....	74
2.4.2 Adware.....	74
2.4.3 Trojan.....	75
2.4.4 Rootkit	76
2.4.5 Botnets and viruses	77
2.4.6 Worms.....	78
2.5 MOBILE MALWARE INFECTION VECTORS	80
2.5.1 Bluetooth/SMS distribution	81
2.5.2 USB/network to device distribution.....	83
2.5.3 Market/application to device distribution	83

2.6 CLASSIFICATION OF SMART MALWARE BASED ON BEHAVIOURAL	84
2.6.1 Classification based on privilege escalation.....	84
2.6.2 Classification based on attack goals.....	85
2.7 CONCLUSION	88
ACKNOWLEDGEMENTS.....	89
REFERENCES	90
CHAPTER 3: ANALYSIS OF ANDROID MALWARE DETECTION TECHNIQUES: A SYSTEMATIC REVIEW	100
3.1 INTRODUCTION	101
3.1.1 Malware	102
3.1.2 Malware attacks on Android phones.....	102
3.2 RELATED WORK.....	103
3.3 METHODOLOGY.....	105
3.3.1 Research questions (RQs).....	105
3.3.2 Criteria for papers selection	105
3.3.3 Justification for the selection criteria	106
3.3.4 Source of data	106
3.4 ANDROID MALWARE DETECTION TECHNIQUES.....	107
3.4.1 Dynamic detection techniques	107
3.4.2 Static detection techniques.....	108
3.4.3 Hybrid detection techniques.....	109
3.4.4 Permission-based detection techniques.....	110
3.4.5 Emulation-based detection	110
3.5 DISCUSSION	111
3.6 EVALUATION	112
3.7 CONCLUSION	114
3.8 FUTURE WORK.....	115
ACKNOWLEDGEMENT.....	115
REFERENCES	116
CHAPTER 4: MALWARE LAB EXPERIMENTAL DESIGN.....	119
4.0 INTRODUCTION	119
4.1 MALWARE LAB SETUP AND TOOLS.....	119
4.1.1 APKInspector.....	120
4.1.2 Androguard	121
4.1.3 Dex2jar.....	121
4.1.4 Tcpdump	121
4.1.5 VirusTotal	121
4.2 NETWORK SETUP	121
4.2.1 The design components of the malware lab	124
4.3 DATASET DEPLOYMENT ENVIRONMENT.....	126
4.4 MACHINE LEARNING METHODS APPLICATIONS IN CYBERSECURITY	129
4.4.1 Phishing detection.....	130
4.4.2 Intrusion detection	130
4.4.3 Breaking CAPTCHAs.....	131
4.4.4 Spam detection on social networks.....	131
4.4.5 Malware detection and classification	132
REFERENCES	133
CHAPTER 5: HOST-BASED DETECTION AND ANALYSIS OF ANDROID MALWARE: IMPLICATION FOR PRIVILEGE EXPLOITATION	136
5.1 INTRODUCTION	137
5.2 LITERATURE REVIEW	138
5.3 METHODOLOGY.....	139
5.3.1 System setup	139
5.3.2 Dependencies	140
5.4 MACHINE LEARNING ALGORITHM.....	142

5.4.1 Experiment.....	142
5.4.2 Malware corpus timeline.....	143
5.4.3 Detection and feature extraction	145
5.5 METRICS FOR PERFORMANCE EVALUATION	147
5.6 EVALUATION	149
5.6.1 Permission request and URLs contacted.....	149
5.6.2 Network communication.....	151
5.6.3 Payload and location monitoring	152
5.6.4 Battery depletion.....	154
5.7 CONCLUSION	154
ACKNOWLEDGEMENT	155
REFERENCES	156
CHAPTER 6: ANDROID PERMISSION CLASSIFIER: A DEEP LEARNING ALGORITHMIC FRAMEWORK BASED ON PROTECTION AND THREAT LEVELS	158
6.1 INTRODUCTION	159
6.2 BACKGROUND AND RELATED WORK.....	162
6.2.1 Similarity feature extraction.....	162
6.2.2 Permission requests and detection algorithms	163
6.3 MATERIALS AND METHODS.....	164
6.3.1 Malware dataset	164
6.3.2 APK file decompilation	165
6.3.2.1 Permission features	165
6.3.2.2 API call features.....	166
6.3.2.3 Command signature features	166
6.3.2.4 Intent features.....	166
6.4 FEATURE SELECTION AND DATABASE GENERATION.....	167
6.5 FEATURE CLASSIFICATION	170
6.6 BINARY FEATURE REPRESENTATION	172
6.7 FRAMEWORK DESCRIPTION.....	178
6.7.1 Network overfitting.....	181
6.7.2 Regularization of feature diversity.....	182
6.7.3 System learning approach	184
6.8 EXPERIMENTAL EVALUATION	185
6.8.1 Experimental setup.....	185
6.8.2 Evaluating performance metric	185
6.8.3 Evaluating feature diversity of the APC model	191
6.8.4 Evaluating permission and threat levels.....	194
6.9 CONCLUSION	195
ACKNOWLEDGMENTS	196
REFERENCES	197
CHAPTER 7: MODELLING CORRELATION BETWEEN ANDROID PERMISSION VARIABLES BASED ON THREAT AND PROTECTION LEVEL USING EXPLORATORY FACTOR PLANE ANALYSIS.....	202
7.0 INTRODUCTION	203
7.1. BACKGROUND	205
7.1.1 Android permission architecture and other components.....	205
7.1.2 Protection levels and permission flags.....	207
7.1.2.1 Normal protection level.....	207
7.1.2.2 Signature protection level.....	207
7.1.2.3 Dangerous protection level.....	207
7.1.2.4 Signature or System protection level.....	208
7.1.3 Intent message.....	208
7.1.4 API calls.....	209
7.2 RELATED WORK.....	211
7.2.1 Permission-based detection and feature extraction	211

7.2.2 Control flow graph.....	212
7.2.3 Bayesian correlation, opcode sequence, and t-Distribution Stochastic Neighbor Embedding (t-SNE) ..	213
7.3 MATERIALS AND METHODS.....	214
7.3.1 Data set	215
7.3.2 The t-Distribution Stochastic Neighbor Embedding.....	215
7.3.3 The Self-Organising Map (SOM)	219
7.3.4 Exploratory factor analysis.....	221
7.3.5 Correlation between Android permissions.....	225
7.3.5.1 Pearson correlation	230
7.3.5.2 Spearman Correlation.....	233
7.3.5.3 Kendall Correlation	234
7.3.6 Comparison of the correlation coefficients	236
7.4 THREAT AND PROTECTION LEVELS EVALUATION.....	237
7.5.1 RESULTS AND DISCUSSION	243
7.5.2 PERMISSION REQUEST AND PRIVILEGES ESCALATION	252
7.6 CONCLUSION	253
REFERENCES	255
CHAPTER 8: GENERAL DISCUSSION	262
8.0 INTRODUCTION	262
8.1 Research question	262
8.2 Research objectives.....	264
8.2.1 Objective set 1.....	264
8.2.2 Objective set 2.....	266
8.2.3 Objective set 3.....	267
8.2.4 Objective set 4.....	268
8.2.5 Objective set 5.....	270
8.3 RESEARCH GENERAL EVALUATION	271
8.4 SUMMARY	273
CHAPTER 9: FUTURE WORK, CONTRIBUTION, AND CONCLUSION.....	274
9.0 INTRODUCTION	274
9.1 OTHER OPEN ISSUES FOR FUTURE RESEARCH	274
9.1.1 Enhancing the security of Android by adding finer granularity.....	275
9.1.2 Trusted applications	275
9.1.3 Malware attacks on RFID technology-based applications	276
9.1.3 Enhancing the security of smartphones through evidence analysis	277
9.2 SUMMARY OF CONTRIBUTIONS	278
9.2.1 Contribution 1: Paper 1	279
9.2.2 Contribution 2: Paper 2	280
9.2.3 contribution 3: Building malware lab	281
9.2.4 Contribution 4: Paper 3	282
9.2.5 Contribution 4: Paper 4	282
9.2.6 Contribution 4: Paper 5	283
9.3 Conclusion	284
REFERENCES	286
APPENDIX A: CERES REQUEST FOR A COPY SERVICE.....	287
APPENDIX B: CORRELATION HEATMAPS	288

LIST OF FIGURES

Figure 1-1 Diagrammatic representation of Flubot malware infection trick	24
Figure 1-2 Worldwide distribution of mobile OS platform in the market share in 2021.....	25
Figure 1-3 Android OS architecture [23].....	27
Figure 1-4 Structure of Android APK file format	28
Figure 2-1 Scientometrics dataset selection, filtration, and elimination process	54
Figure 2-2 Determining the total increase in node purity of the most important keywords in the Topic Search from the combined dataset.....	55
Figure 2-3 Path diagram showing the keywords characterization and the number of factors using simplimax as the rotation method and minimum residual as the estimation method.....	56
Figure 2-4 Comparison of malware attack rates on mobile OS platforms in 2013 [38].....	59
Figure 2-5 Illustrating mode of operation of NOtCompatible.C Android malware	65
Figure 2-6 Chamois backdoor operational propagation stages with each encrypted and obfuscated	70
Figure 2-7 Comparison of PHA categories distribution outside of Google Play in 2017 and 2018 [75]	70
Figure 2-8 Mobile virus infection strategy	78
Figure 2-9 Malware propagation using Bluetooth technique	82
Figure 2-10 Mobile malware Privilege escalation illustration.....	85
Figure 2-11 Mobile malware characterization.....	87
Figure 4-1 Setting VMs Ip address to host-only-adaptor	123
Figure 4-2 VM structure.....	124
Figure 4-3 Integrated multi-Personal Computer in our malware lab	125
Figure 4-4 Virtualisation environment for the VM architecture of the hypervisor controlling component.....	126
Figure 4-5 Malware deployment architecture of the lab design	128
Figure 5-1 System setup for detection and analysis	140
Figure 5-2 Distribution of the Android malware corpus using Engauge digitizer.....	143
Figure 5-3 Authentication and detection	145
Figure 5-4 Classifying benign and malicious applications at each iteration.....	146
Figure 5-5 ROC curve showing classification accuracy.....	148
Figure 5-6a Permissions requested	150
Figure 5-6b URLs contacted, illustrating potential privileges exploited by Fonefee.b android malware.....	150
Figure 5-7 Files accessed by fonefee.b malware	151
Figure 5-8 Malware activity connectivity.....	152
Figure 5-9 Fonefee.b Payload.....	153
Figure 5-10 Location monitoring	153
Figure 6-1 The overall system architecture	180
Figure 6-2 An illustration of the neural network working operation function of the proposed classifier.....	182
Figure 6-3 ROC curve showing the AUC of the classifier	186
Figure 6-4 Feature diversity of the APC framework compared with existing classifiers	188
Figure 6-5 Effects of threshold and diversity factor on feature regularisation	190
Figure 6-6 Dangerous permissions classification by the framework.....	193
Figure 6-7 Classified permissions with different threat levels by our framework.....	194
Table 7-1 Android code name release versions show the increase in the API level and number of permissions.....	206
Figure 7-1 Protection level and permission-based flags.....	209
Figure 7-2 Android OS API call architectural handling process showing the connection between the APIs structure and Call states interfaces.....	210
Figure 7-3 Gaussian Kernel optimisation values.....	218
Figure 7-4 Points correlation and proportions	218
Figure 7-5 SOM weight positions of the permission input variables	220
Figure 7-6 SOM U-matrix for Android permissions. The hexagons indicate spaces between network neurons with a binary string 0, 1 which forms the Euclidean distance	220
Figure 7-7 The Scree plot shows how much there is variance in the dataset by eigenvalues in the three factors	221
Figure 7-8 Exploratory analysis path diagram of the correlation factors and the permission variables	223
Figure 7-9 Exploratory visualisation of permission	225
Figure 7-10 Density plot for random permission variable.....	227
Figure 7-11 Sequential analysis of the empirical evidence of dangerous permission with high threat levels.....	229

Figure 7-12 Flag supported correlation when permission distribution is heterogeneous in the sample distribution	231
Figure 7-13 Pearson's correlation coefficient for some selected permission variables	232
Figure 7-14 Spearman's rho correlation coefficients	234
Figure 7-15 Kendall's tau-b correlation coefficient for some selected permission variables	235
Figure 7-16 Correlation at different degrees of coefficient strengths	236
Figure 7-17 Model posterior distribution showing the correlation between the observed data (protection level) and the inferred data (threat level)	240
Figure 7-18 Plot Matrix showing exploratory plane analysis visualization of the correlation between threat and protection level in permissions	241
Figure 7-19 Exploratory factor plane visualization of permissions threat and protection level class correlation	243
Figures 7-20 Boxplots showing correlation between protection and threat level of some selected dangerous permission variables	244
Figure 7-21 Scatter plots showing the protection and threat level of each permission per count	245
Figure 7-22 Visualisation of some sensitive API showing their path index, threat level, caller code, and operation	246
Figure 7-23 Classifying protection and threat level of certain permissions based on their usage	247
Figure 7-24 Android applications threat level obtained by our model	248
Figure 7-25 Cluster density plots showing dangerous permission-based clustering	249
Figure 8-1: Research methodology demonstrating how the chapters are connected in this thesis	263

LIST OF TABLES

Table 2-1 Mobile malware incidents and evolution covering 2000 to 2020	60
Table 2-2 Android malware evolution in the period of 2010-2020	72
Table 2-3 Mobile malware variants with their distinctive characteristics	80
Table 2-4 Mobile malware infection objectives	88
Table 3-1 Comparative analysis of the studied works	113
Table 5-1 LMD rules	141
Table 5-2 Dataset timeline consisting of Android malware families within the sample files	144
Table 5-3 Confusion matrix	148
Table 6-1 Cluster containing feature vector binary attributes	168
Table 6-2 Data set attributes distribution	168
Table 6-3 Attributes logical table	169
Table 6-4 Decision list using associated rules	171
Table 6-5 Intent feature vectors	173
Table 6-6 Commands signature feature vectors	174
Table 6-7 Permission feature vectors	174
Table 6-8 API call signature feature vectors	176
Table 6-9 Performance metrics comparison	187
Table 6-10 Performance comparison with traditional classifiers	187
Table 7-1 Android code name release versions showing the increase in the API level and number of permissions	206
Table 7-2 Structure Matrix of some selected variables using equamax rotation method	222
Table 7-3 Interaction effects of the test statistics using MANOVA	228
Table 7-4 Posterior estimate of the model	231
Table 7-5 Evaluation metrics	240
Table 7-6 Correlation model comparison of some selected permissions	251

LIST OF EQUATIONS

(6-1)	170
(6-2)	170
(6-3)	171
(6-4)	172
(6-5)	172
(6-6)	172
(6-7)	172
(6-8)	177
(6-9)	177
(6-10)	177
(6-11)	179
(6-12)	179
(6-13)	179
(6-14)	179
(6-15)	179
(6-16)	179
(6-17)	179
(6-18)	179
(6-19)	180
(6-20)	180
(6-21)	180
(6-22)	181
(6-23)	181
(6-24)	181
(6-25)	183
(6-26)	183
(6-27)	184
(7-1)....	216
(7-2)	216
(7-3)	216
(7-4)	217
(7-5)	217
(7-6)	219
(7-7)	219
(7-8)	219
(7-9)	226
(7-10)	226
(7-11)	226
(7-12)	227
(7-13)	228
(7-14)	228
(7-15)	228
(7-16)	229
(7-17)	229
(7-18)	229
(7-19)	230
(7-20)	233
(7-21)	234
(7-22)	238
(7-23)	238
(7-24)	238

(7-25) 239

(7-26) 242

(7-27) 242

(7-28) 242

(7-29) 242

ABBREVIATIONS

ABT	Ada Boosted Tree
ACPO	Association of Chief Police Officers
API	Application Program Interface
A2D	Application to Device
APK	Android Application Package
APC	Android Permission Classifier
ARFF	Attribute Relation File Format
AUC	Area Under Curve
AV	Attack Vectors
BR	Bayesian Regularisation
C0&C1	Cluster labels
CD	Core Data
CDT	Coarse Decision Tree
C&C	Command and Control server
CFG	Control Flow Graph
CSV	Comma Separated Values
CSVM	Cubic Support Vector Machine
CVV	Card Verification Value
CP	Cross Platform
DFRWS	Digital Forensic Research Workshop
DIU	Digital Investigation Unit
DNN	Deep Neural Network
DDoS	Distributed Denial of Service
DoS	Denial of Service
DVM	Dalvik Virtual Machine
ED	Extended Data
ELF	Executable and Link Format
FN	False Negative
FNR	False Negative Rate
FP	False Positive
FPR	False Positive Rate
FR	Feature Regularisation
FSA	Finite State Automata
FVG	Finite Volume Grid
GPS	Global Positioning System
HMMs	Hidden Markov Models
IBK	Instance Based Learner

ID	Identity Document
IDC	International Data Corporation
IG	Information Gain
IMDs	Implantable Medical Devices
IMEI	International Mobile Equipment Identity
IMSI	International Mobile Subscriber Identity
JAR	Java Archive
J2ME	Java 2 Micro Edition
KNB	Kernel Naïve Bayes
LMD	Linux Malware Detect
MIM	Malware Infection Model
MMS	Multimedia Messaging Service
MS	Multiple Scale
mTAN	Mobile Transaction Authentication Number
NFC	Near Field Communication
N2D	Network to Device
NHS	National Health Service
NCSC	National Cyber Security Centre
NIJ	National Institute of Justice
NPL	Natural Language Processing
OS	Operating System
OEM	Original Equipment Manufacturers
PCA	Principal Component Analysis
PDA	Personal Digital Assistant
PDF	Probability Density Function
PID	Process Identifier
PIN	Personal Identification Number
POI	Point of Interest
PHAs	Potential Harmful Applications
PVM	Process Virtual Machine
QEMU	Quick Emulator
RATs	Remote Access Trojans
RFID	Radio Frequency Identification
ROC	Receiver Operating Characteristic
SDK	Software Development Kit
SIM	Similarity Measure
SMS	Short Message Service
SOA	Service Oriented Architecture
SVMs	System Virtual Machines

SVM	Support Vector Machine
TN	True Negative
TP	True Positive
UPX	Ultimate Packer for Executables
UW	User Wanted
VM	Virtual Machine
WEF	World Economic Forum
WKNN	Weighted Kernel Nearest Neighbour
XML	Extensible Markup Language

Prologue

We present this thesis as a sequence of papers that are either published or in preparation for publication. In each of the papers, a related review of the literature was provided as a contribution to provide a background for placing distinct papers and the entire work. Also, at the end of each paper, references are provided to show the context to which the object is connected to the existing related work. We present the summary of the chapters of the thesis papers as shown in the table below.

Chapter	Name of paper	Journal	Status
Chapter 2	Android Mobile Malware Evolution and Infection Strategies: A Survey	International Journal of Security and Privacy	Published (January/2022)
Chapter 3	Analysis of Android Malware Detection Techniques: A Systematic Review	International Journal of Cyber-Security and Digital Forensics (IJCSDF)	Published (September/2019)
Chapter 5	Host-Based Detection and Analysis of Android Malware: Implication for Privilege Exploitation	International Journal for Information Security Research (IJISR)	Published (June/2019)
Chapter 6	Android Permission Classifier: A Deep Learning Algorithmic Framework Based on Protection and Threat Levels	International Journal of Security and Privacy	Published (May/2021)
Chapter 7	Modelling Correlation Between Android Permission Variables Based on Threats and Protection Level Using Exploratory Factor Analysis	Journal of Cybersecurity and Privacy	Published (Nov./2021)

CHAPTER 1: INTRODUCTION

1.1 Malware attacks and economic impacts

If malware infects a digital device, then it has control over the device's functionality and data. This can impact your finances, your privacy, and your access to your data. Malware is a threat not only to individuals but also to corporate organisations and financial institutions as well. Malware attacks could lead to communication traffic of an infected network, hardware failure of the physical device, data theft, and loss of critical business data, among others. The extent of damage malware causes is dependent on whether the effect is on the corporate organisation or individual users. While the impact of the malware infection may be serious, the damage in other cases may be unnoticeable and less severe. Also, the impact of the harm may differ on the kind of malware, the type of data accessed or stored by such a device or infrastructure, and the nature of the device infected. As a result, there is a need to improve malware detection techniques to minimise rates of attacks and infection on devices and network infrastructure.

Among other digital devices, mobile devices have been under severe threat more than ever before due to mobile technological advancements. The advancement in mobile technology such as the 5G network which accelerates IoT device connectivity; advancement in Bluetooth technology, cloud computing, and smartphone internet connectivity, and many more have integrated the world's entities and places which has brought ease to the human lifestyle [1]. However, these advancements have made mobile security face different forms of threats such as phishing, malware attacks, and data leakage and loss, among others. Some recent cases of malware attacks are highlighted in the subsection.

1.1.2 Recent cases of malware attacks

In June 2019, Lake City was attacked by ransomware which for two weeks crippled the computer systems operation of the county. A ransom worth \$500000 was paid to the hackers via the insurance provider of the city government council. This malware attack was similar to the one that cost Riviera City and Jackson County that paid \$600000 and \$4000000 ransom respectively. According to Cristea [2], malware attacks on digital devices is one of the most challenging security issues both at national and international context. While the ransom attack on Riviera City and Jackson County was generic, the Lake City ransomware encrypted email systems and locked mobile phones. In 2019, the City Power in South Africa suffered a horrific ransomware attack that caused severe disruption in the electricity supply. The malware barred customers from accessing the electricity distribution company's website where they can buy electricity units. The attack left prepaid customers without electricity for a couple of weeks [3] resulting in economic inactiveness. Capital One is one of the victims who suffered a severe data breach and financial loss when hit by ransomware [4]. More than 106 million credit card applicants were affected by the malware attack. The attack was pre-coordinated which affected more than 20 organisations in Texas. A ransom of \$2.5 million was paid to retrieve the encrypted data. This resulted in both financial loss and the operational shutdown of the affected organisations.

While the first quarter of 2019 experienced a 50% cyber-attack increase compared to 2018, the year 2020 witnessed a distinctive increase in the cyber threat with malware becoming the utmost significant threat [5]. According to [6], the COVID-19 pandemic constitutes an attack vector in the world of cyber where remote working and work from home approaches initiated by many countries to reduce infection became a necessity. This has enabled threat actors to exploit the uncertainties resulting from the pandemic which are infrastructural and technologically oriented. Cybercriminals began to adapt swiftly to the global pandemic to integrate the theme in their malware attack lures process. Some of the pandemic themes included but were not limited to contact tracing apps, free mask availability news, vaccination app, and vaccine news, among others. These themes and much more play significant roles in cybercrime including phishing, social engineering, and malware infection landscape all through 2020 and beyond. While organisations continue to deploy different work strategies and standards such as VPN endpoint deployment and online tools collaboration, the attack surface continues to increase [7].

In 2020, Manchester United football club was hit by a cyber-attack. The attack was discovered to be ransomware where the IT infrastructure of the club was disrupted for two weeks. The hackers demanded ransom [8] which could result in the club's data and financial loss as a consequence of shutting down the revenue generation from its operation. Cyber-attacks continue to surge in 2020. In 2021, a significant number of malware attacks on both public and private organizations have been witnessed so far. The National Cyber Security Centre (NCSC) reported that UK educational sector was heavily attacked by ransomware in February 2021. According to the report, 'in recent incidents affecting the education sector, ransomware has led to the loss of student coursework, school financial records, as well as data relating to COVID-19 testing' [9]. The attack caused an overwhelming impact on the educational industry, with significant financial loss and time to recover and restore the critical services.

In their recommendation, while responding to the attacks, NCSC highlighted common infection vectors including unpatched devices, phishing, and permission privilege escalation, and lateral movement, among others. While providing malware and ransomware mitigation strategies, NCSC emphasized the need to develop application permission request techniques that could centrally manage organisational and personal devices. Training and providing awareness on permitting only applications from the trusted market and limiting excessive granting of permission access was emphasised [10] as one of the ways to prevent malware attacks on individual and public infrastructures. While the number of malware attacks keeps surging generally, attacks on mobile platforms such as Android were more prevalent.

On the 22nd of May 2020, an internet security company (ESET) announced a new android mobile malware called DEFENSOR ID that clears victims' bank account and takes over social media and email accounts [11]. DEFENSOR ID is a banking Trojan that has information-stealing abilities which require only single permission from the victim during installation to perform its malicious actions. This abused the permission and accessibility services offered by Android resulting in the stealing of sensitive data and the reading of displayed text messages received by the victim. DEFENSOR ID modified the application protocol of the infected devices leading to firebase cloud usage for messaging. This is similar to the Android malware attack of 2019 called 'Agent Smith', which infected 25 million Android users worldwide [12]. The malware was concealed within the WhatsApp application during installation, thus, taking advantage of the Android OS vulnerabilities

such as pileup flaws. Agent Smith affected 15 million victims in India, more than 300,000 in the U.S., 137,000 in the U.K., and the remaining victims in other countries.

Malware attacks on corporate organisations and mobile infrastructures keep evolving from simple to more sophisticated attacks. While mobile devices were directly targeted, they were also used as zombies to perform malicious operations remotely. This is seen in the case of EasyJet Airline Company in April 2020 where 9 million customers were affected. About 2,208 credit and debit card details including digital security code (CVV number) were stolen by the attackers. The stolen data was later used for phishing to further lure customers to supply more personal details through malicious emails and mobile apps [13].

On the 29th of April 2021, a mobile malware called Flunot infected more than thousands of users of Android devices. The malware infected victims through text messages received by the user [14-15]. The text tricked thousands of mobile users, especially those using Android-based platform OS into believing that the victim has missed delivery of his parcel from branded companies such as Amazon, DHL, Royal Mail, Hermel, and UPS. According to the report, Flubot provides an app download link where the victim can trace and reclaim the missed delivery parcel. The provided link is spyware which gains access to the victims' device via permissions access and steals sensitive information such as banking details, passwords, among others. According to the report provided by the UK National Cyber Security Centre [16], Flubot infected thousands of Android users but with a low current threat on Apple device users. Figure 1-1 demonstrated the Flubot malware's infection strategy as described by the UK National Cyber Security Centre.

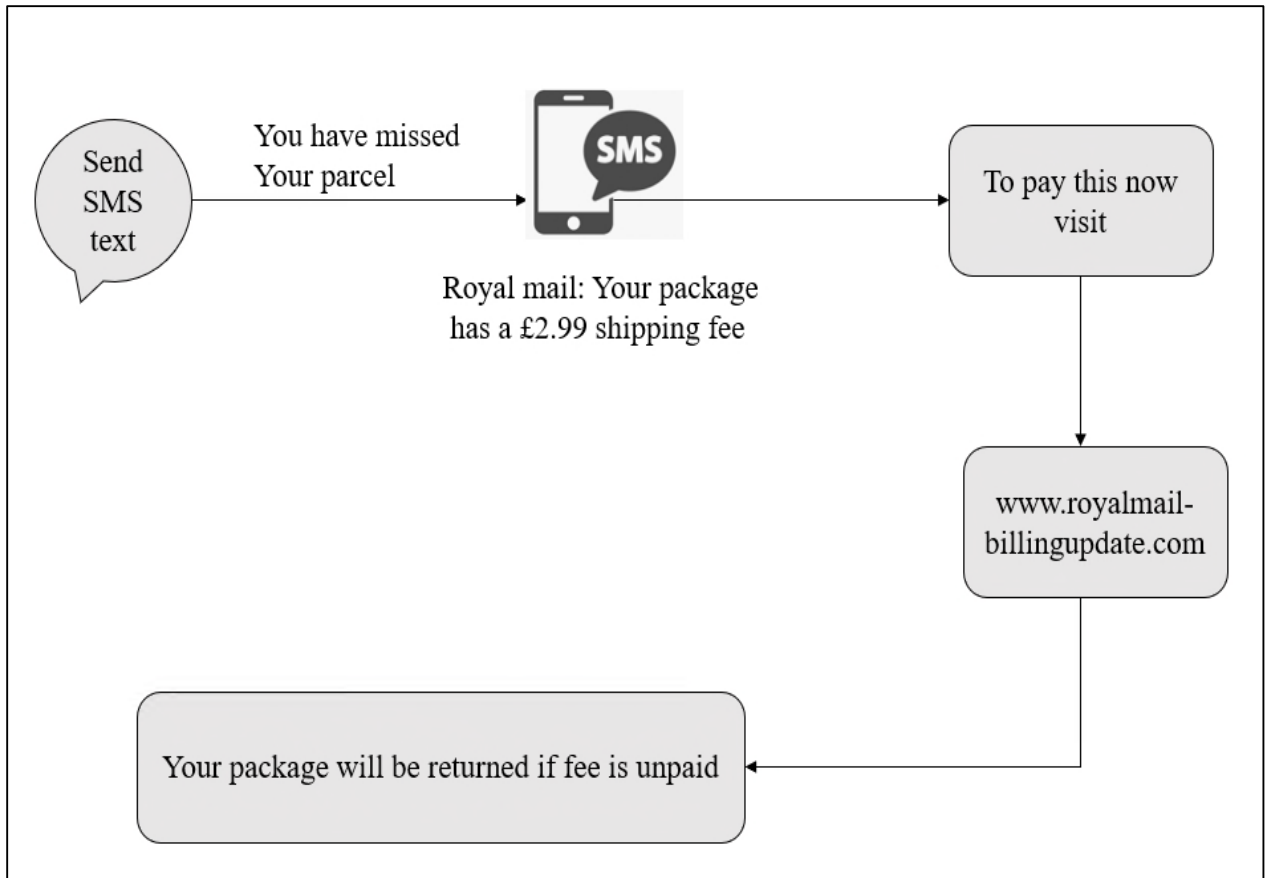


Figure 1-1 Diagrammatic representation of Flubot malware infection trick

1.2 Mobile Devices

Mobile devices are used more and more for a predominant portion of our day-to-day lives. This is due to their exceedingly important computational functionalities and services such as online banking, mobile banking, email applications, and online shopping, among others. The computational capabilities of mobile devices are characterised by fast mobile technological advancements which influenced mobile users [17]. A recent study [18] reported that 5.19 billion people use different mobile device operating system platforms out of the world's total population of 7.75 billion, with 4.54 billion mobile devices actively connected to the internet. According to Statcounter, Android OS has a 72.72% share in the mobile OS market worldwide in 2021 as the most popular mobile OS [19] as compared to other mobile OS such as iOS, Windows, Linux, among others (as shown in figure 1-2). The leading factors for Android operating system

popularity are connected to its open-source platform nature. Other factors such as easy usage and low cost also influence its popularity. For instance, unlike iOS, Android phones are relatively cheaper. The popularity of Android increases the number of attacks targeting the OS. Laughlin [20] reported that over one billion Android-based devices are infected with different malware variants as of 2020.

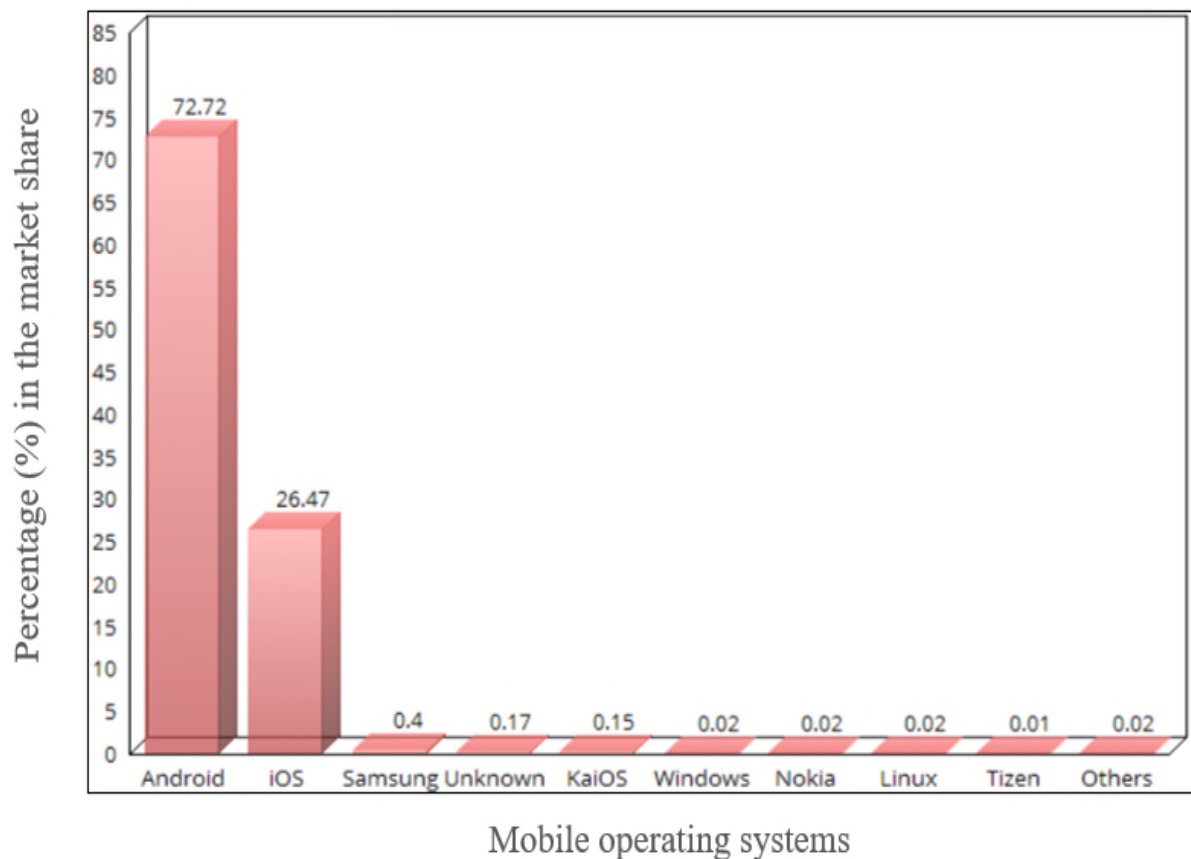


Figure 1-2 Worldwide distribution of mobile OS platform in the market share in 2021 [19]

The fast growth of Android has led to an increase in interest from hackers, because of the sensitive information they collect from many victims. Many android users use their mobile devices to perform transactions such as mobile banking which includes activities such as credit and deposits alerts, account threshold, and loan statements. Since hacking into financial and organisational websites can be forensically detected, black hackers, deploy other subtle ways to evade detection, stealing victims' details by using malicious applications. These are programs written by individuals or a group of persons to perform malicious activities on the victim's device or other infrastructures. It is, therefore, significant to investigate strategies that hackers deploy to infect Android OS with malware by studying the architecture of the Android system.

1.3 Android operating system

In 2008, at its inception, the Android operating system was called Alpha [21]. Android versions keep evolving with different code names and API levels (Table 1-1). For example, in 2014, the latest Android name is Lollipop with a 21-23 API level, higher versions such as nougat emerged in 2016, Oreo version in 2017, and the latest beta 1 [22]. Though different versions keep emerging, the Android operating system composition (figure 1-3) remains the same.

Table 1-1 Android version evolution[21][22]

Code name	Version number	Released date	API Level
Alpha	1.0	Sept 23, 2008	1
Beta	1.1	Feb 9, 2009	2
Cupcake	1.5	April 27, 2009	3
Donut	1.6	Sept 15, 2009	4
Éclair	2.0-2.1	Oct 26, 2009	5-7
Froyo	2.2-2.2.3	May 20, 2010	8
Gingerbread	2.3-2.3.7	Dec 6, 2010	9-10
Honeycomb	3.0-3.2.6	Feb 22, 2011	11-13
Ice Cream	4.0-4.0.4	Oct 18, 2011	14-15
Jellybean	4.1-4.3.1	July 9, 2012	16-18
KitKat	4.4-4.4.4	Oct 31, 2013	19-20
Lollipop	5.0-5.1.1	Nov 12, 2014	21-23
marshmallow	6.0-6.0.1	Oct 5, 2015	23
Nougat	7.0-7.1.1	Aug 22, 2016	24-25
Oreo	8.0-8.1	Aug 21, 2017	27
Android Pie	9	Aug6, 2018	28
Android Q	10	March 13, 2019	29
Android 11 beta	11	June 10, 2020	30

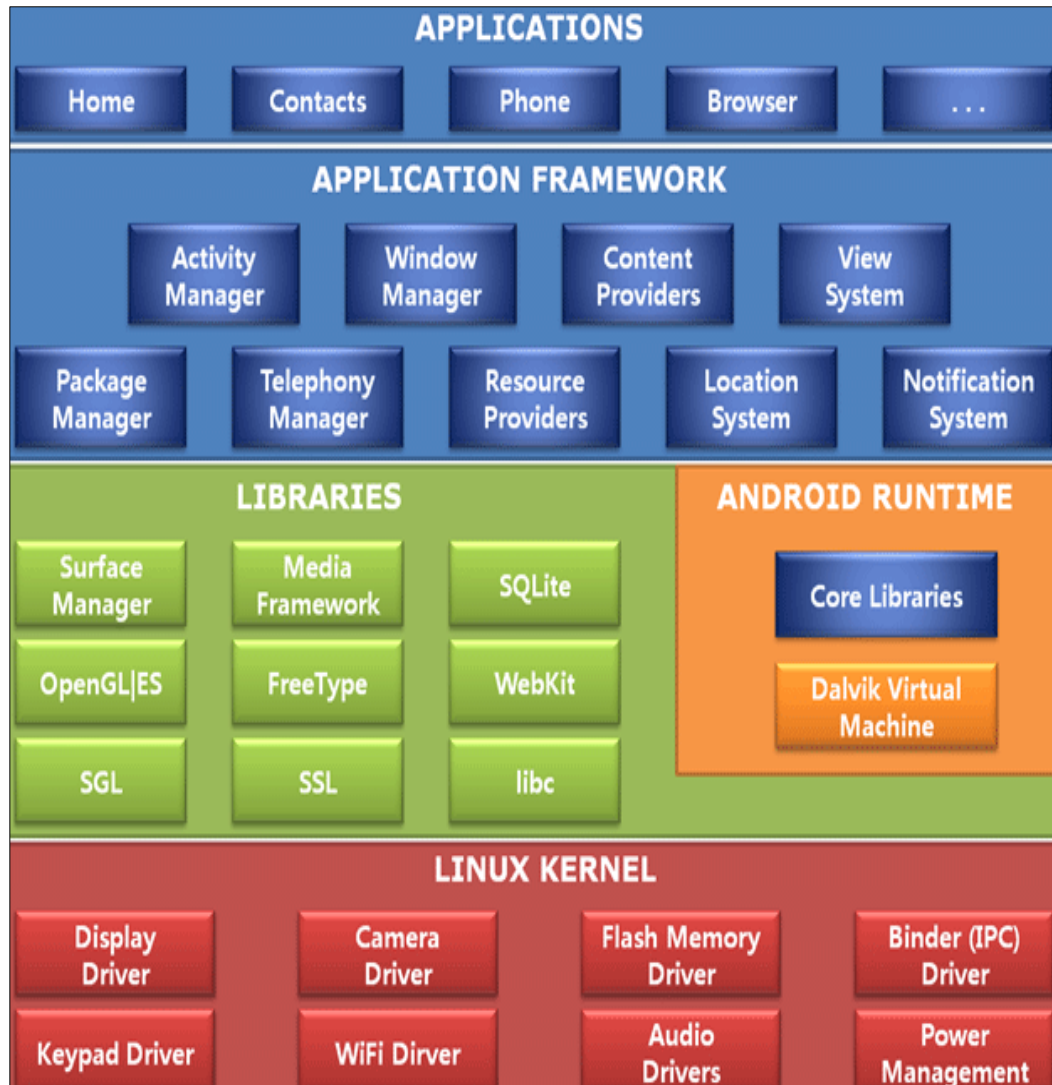


Figure 1-3 Android OS architecture [23]

1.3.1 Structure of APK file

Android Application Package (APK) [24] enables application distribution between mobile devices and market providers. APK is a well-organised package with Java ARchive (JAR) formatting style housing different Android files such as executables, classes, manifest, raw resources, and library. Malware applications are also encapsulated in this file formatting in the same manner by hackers. APK is used by Android for application installations and building of other dependencies and frameworks respectively. To build a classification framework that will be robust in identifying features of a malicious file, we examined APK structure (as shown in figure 1-4) to understand

features related to each segment. For instance, the manifest.xml contains all the permission variable attributes which most malware variants exploit to infect the victim's device. This research, therefore, provides a detailed architectural representation of APK file format structure looking at all the connecting components and services. When an APK file is decompiled using any tool such as the JADX compiler, basic structures of the file are extracted for further analysis of the application's structure. Understanding the structure of an APK is significant for analysing Android-based malware.

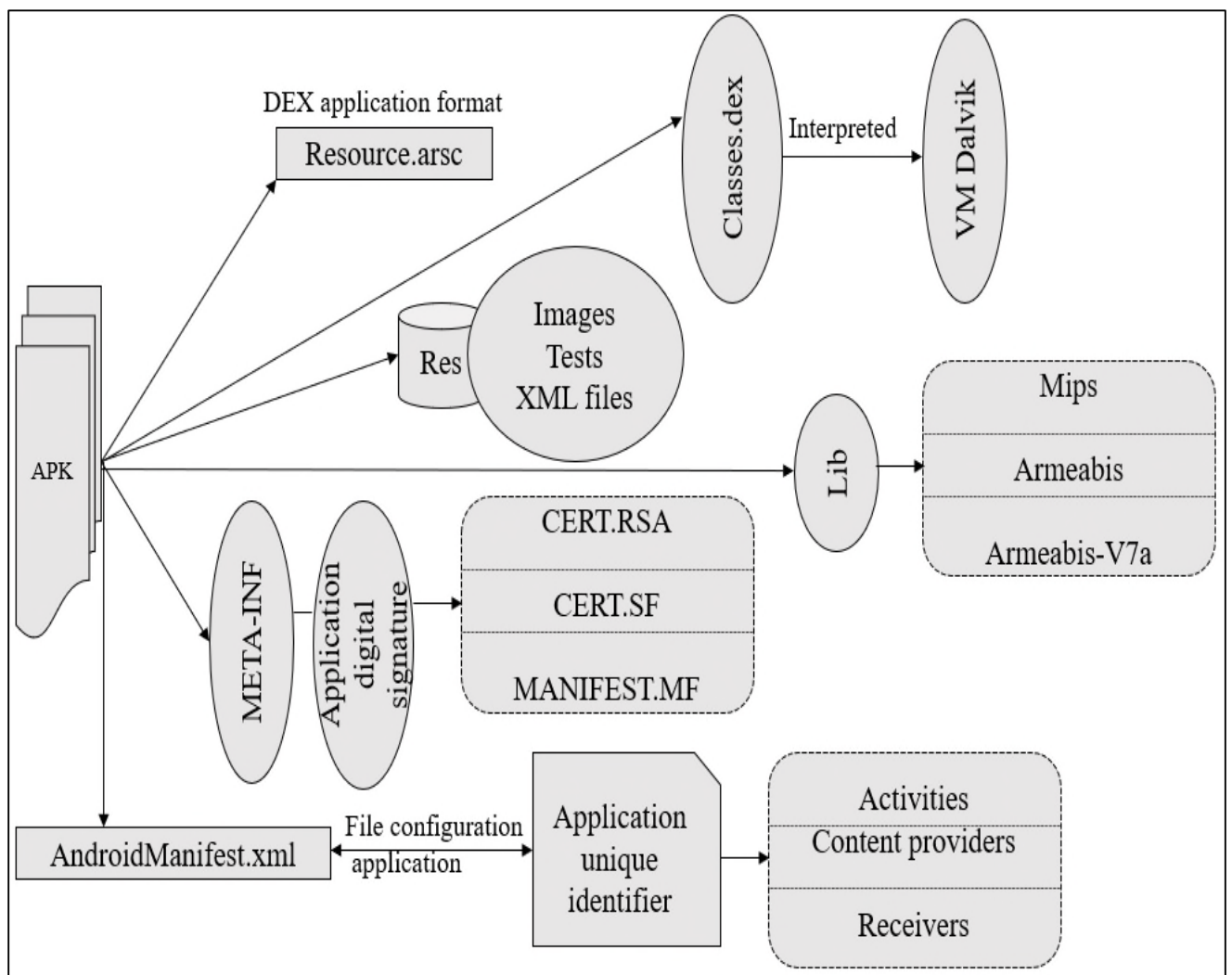


Figure 1-4 Structure of Android APK file format

1.3.2 Taxonomy of APK structure

The basic structure of an APK contains assets, lib, original, res, resources, and sources configuration files.

- **Assets:** It is a segment of the APK configuration which contains filenames for permissions privileges and methods used to request for them respectively. Principally, data3, data1, data4, data2, fj, and fs are the hierarchical arrangements of the file structure. Details of malicious activities can be investigated through analysis of these file sectors.
- **Lib:** Armeabi folder of this file structure creates a floating contact for application debugging of readable and re-writeable data of Libf-jni.So and libdata.So respectively. It is a library in the Android environment module that facilitates the management of operational services at the OS background of the Android. This configuration structure is composed of mips, armeabis, and Armeabis-V7a.
- **Original:** This is where META-INF and AndroidManifest are located. Other important components of the original archive tree are ANDROID_SF, ANDROID_RSA and MANIFEST.MF respectively. It is an associate sector of the resources file.
- **Resources:** This contains two primary features which are a folder called res and AndroidManifest.xml. Android drawable, values, layout, anim, and layout are controlled by the res component. In Android architectural development, res represents sets of defined Android resources that an application can have access to. These resources range from images, textual information, XML files, colours, values from strings and menus. It encloses application arrays, styles, strings, dimensions, and ids that are rooted in an XML file format. In identifying attack vectors that malware deploys to infect android based platforms, understanding the essential components of the res file segment is critical. Malicious applications such as adware and pop-up Ads can contravene Android security and sediments their payloads by infecting the online_pop_up sector of the Android application resources. Adequate knowledge of how the APK structure of Android works will facilitate the understanding of permission request attributes by malicious applications.

1.4 Digital investigation and mobile malware

Technological growth with the high usage of digital devices has created an emerging field of computing called digital forensics. As technology is digitally growing so does digital crimes. The word forensics originated from the Latin word forensic which means forum [25]. It is a field of science that is concerned with investigating crimes and pieces of evidence related to digital devices. Dogan and Akbal [26] defined digital forensics as a science which deals with analyzing electronic proofs which are retrieved from digital devices. This definition lacks some fundamental details of what digital forensics is all about. Critical forensic investigation processes, phases, and components such as detecting, extracting, preservation, presentation, and evidence repeatability and reporting were not considered. The definition covered less than 30% of the digital forensics' life cycle.

The research of [27] defines digital forensics as a branch of forensics that deals with the production, transmission, and storage of computer evidence for legal exercise. The research affirmed that most electronic devices carry a massive amount of evidence which when extracted can be of evidential value to legal practitioners and forensics investigators at large. The definition, however, could not unveil the totality of what electronic evidence entails. Other sources of digital evidence such as mobile devices, smartwatches, smart TV, smart fridge, deep freezers, and other areas were not captured in their definition and concept. The most commonly quoted and acceptable definition is the one given by the Digital Forensic Research Workshop (DFRWS) which defines digital forensics as 'The use of scientifically derived and proven methods toward the preservation, collection, validation, identification, analysis, interpretation, documentation, and presentation of digital evidence derived from digital sources to facilitate or further the reconstruction of events found to be criminal, or helping to anticipate unauthorized actions shown to be disruptive to planned operations'[28-29].

The definition given by the DFRWS in 2001 is the most complete as all the segments of digital forensics (DF) are considered. DF can be seen in simple terms as applying all the forensically sound methods to ensure that all the original evidence is not altered but preserved. This can be equated to general crime scenes where law enforcement has standardized some investigative principles and approaches that must be applied and strictly followed as the basis when dealing with evidence. Evidence worthy to be presented and accepted in the court of law must not be captured haphazardly but carefully and logically.

Digital evidence also called electronic evidence is any supportive data in transit or storage either on an electronic device or storage media that could be used to negate or back up a criminal-related theory or investigation. Evidence stored on digital devices is highly delicate. As a result, their collection and preservation process must be handled with a high mark of carefulness from acquisition, analysis, and reporting to maintain evidence integrity. Digital forensics evidence must be unaltered for it to be admissible and acceptable when presented in the court of law. Evidence must be relevant to the case, permissible legally, inherently reliable in the court, identifiable and preservative, and must not be modified.

Digital evidence could be found in different sectors of the device or storage media both in allocated and unallocated space. Evidence could be sourced also from suspicious accounts, applications, open sockets, and hidden files. Digital evidence could be artefacts in form of phone contacts, events timelines, messages, thumbnails, encrypted files by ransomware, etc. Understanding these principles assisted the author on how to handle malware files during the research to avoid escalation. Digital investigation concepts and principles are essential in the investigation and analysis of different properties of malware to identify the attack and sometimes the attack motives.

1.5 Motivation

In this thesis, we recognised three prime issues that need to be researched. The open issues are: There is more mobile malware now targeting Android-based devices platform than ever before. Mobile malware is increasingly becoming sophisticated daily, and most of the existing techniques need improvement to detect smart and other sophisticated malware variants.

- **Open issue 1: There is more mobile malware now targeting Android-based devices platform than ever before.**

As notated earlier (**Section 1.2**), factors such as the open nature of the Android OS, the population using Android devices, and the potential amount of information that can be obtained from the target devices both in storage and on transit has made malware a global pandemic [30] with huge business profit. Other factors such as the open market in the dark web where malware development kits are transacted facilitate the increase of malware. The open-source architecture of the Android OS platform has made millions of applications uploaded on the marketplace with minima security

checks. Consequently, malware attacks particularly on Android-based devices keep rising and have become one of the most security threats been faced by mobile security as highlighted by Joldes [31]. As a result, developing new techniques or enhancing existing ones to improve malware detection to prevent mobile infection is essential.

The research of [32-33] asserted that malware attacks on Android are one of the four most persistent security threats been faced by the mobile community. Due to the increased growth, reuse, and modification of both benign and malicious applications, it has become very difficult to manually perform effective malware analysis. Mobile marketplaces are daily flooded with millions of mobile applications both benign and malicious. Identifying which of those applications are malicious or benign is very challenging and time-consuming especially if the analysis is to be driven manually. This has diminished trust in mobile applications. Looking at the current developmental trend of malware infection on Android, there is an increased infection rate which calls for investigation into their deployment trajectories. Hence, the necessity to have an automated technique to aid intelligent analysis of Android malware is essential.

- **Open issue 2: Mobile malware applications are becoming smart and sophisticated.**

To achieve a successful attack without being detected, new generational malware has become smart and sophisticated in its operation [34]. The new generation of malware variants such as polymorphic and metamorphic malware emerged with different levels of sophistication. The sophistication level of malware is brought about as malware authors keep inventing new evasion techniques. Different evasion techniques like code obfuscation, registry modification, environmental awareness, code transformation, and code encryption and decryption are used by malware writers. Hackers deploy these techniques to enable malicious codes to stay stealthy on the infected device and remain undetected.

The smartness and sophistication of malware generate complications and difficulties during analysis by digital forensic investigators and cyber-security analysts [35]. The behaviour of these smart variants of malware is by far complex to understand and detect when compared to traditional malware which is not characterised by certain features such as the victims' location and profile. This open issue has integrated the necessity for a smart framework that can support the detection and analysis of sophisticated malware based on its functionalities and feature attributes. Understanding the privacy preferences of the users that smart malware leverage via permission

request to access and leak sensitive data of the victim including contacts, SMS, profile, and location are essential in helping to curb sophisticated malware propagation and enhance detection.

- **Open issue 3: The existing techniques need improvement to detect smart malware**

There are several techniques used to detect malware on smartphones such as signature-based which includes the use of anti-virus programs [36] and anomaly-based approaches [37] which include the static [38-39] and dynamic analysis [40-41]. However, the existing techniques have some limitations that need improvement. For example, signature-based detection approaches such as antivirus software are limited in detecting zero-day, polymorphic, and metamorphic malware which are sophisticated and complex in their characteristics and operation. They can only detect malicious variants whose hash and byte signatures are found in the signature repository. In addition, the signatures are designed by humans and as a result, signature-based detection is prone to errors as highlighted in the work of Akhtar [42]. Abnormality-based detection techniques also have limitations in detecting malware including susceptibility to mimicry attacks, the problem of imbalance, and the inability to detect obfuscated malware.

Consequently, smart malware cannot be easily detected and analysed by mere pattern matching as is the usual case with most existing approaches. Even when antivirus software identifies a malware infection on a device, it does not provide critical analytical details of how it invaded the system. As a result, there needs to be a more intelligent framework that could help confront this menace rather than signature harmonization. The security threats highlighted above constitute the open issues that need for designing a robust framework for detecting sophisticated malware.

1.6 Research question

The primary aim of this research is to design a framework that allows for the detection of malicious behaviour of smart malware based on permission attributes performed on the Android OS system. Based on monitoring permission request attributes, a process performing malicious behaviour should be detected by identifying the attack vectors and privileges that smart malware uses to attack Android OS-based devices.

The research question is thus formulated as:

Is it possible to develop a framework that identifies attack vectors and privileges on the permission request system that smart malware deploys to infect Android-based platforms, to prevent malware attacks on Android OS devices?

Recurring patterns and correlations in the permission request system that define malicious behaviour are key research aspects with respect to answering the research question. The sub-questions formulated to answer the principal research questions are:

1. What are the current malware detection techniques (in the literature)? Answering this question provides relevant direction for a new and improved malware detection method based on the techniques examined in the literature review.
2. How can all permissions requested by processes be monitored and classified? Answering this question provides an understanding of the privilege smart malware exploit during permission request processes to exploit and attack Android OS devices.
3. What is a framework to analyse permission requests for anomalies? An effective analysis framework is essential for classifying permissions requests and finding anomalies in the classified system permission attributes of malicious processes.
4. How can the correlation of the permission attributes be defined by threat and protection levels as fundamental classification rules? Ultimately, understanding the correlation between Android permissions will improve the detection and thus prevention of malware on Android OS-based platforms. This framework can be adopted and implemented by anti-malware solutions.

The next section (Section 1.7) outlines the objectives that formed the approach taken to answer the research questions. To provide clear understanding, the following words: Framework, attack vectors, Infection, Android Devices, neural networks were defined for this thesis as follows:

Framework: This is a process that can be built into a module for identifying device-based sophisticated and smart malware to boost the security of Android devices. One of the key characteristics of the framework is the identification and classification of feature attribute that smart malware use to infect Android devices. This framework is significant because it integrates both artificial intelligence components and signature-based techniques for detecting zero-day and sophisticated malware such as polymorphic malware that mutates its code to eschew detection. The details on some of the detection techniques are discussed in chapter 3 section 3.4 subsections 3.4.1, 3.4.2, 3.4.3, 3.4.4, and 3.4.5 respectively.

Attack vector: This is a way or method through which malware penetrates and has unauthorized access to an Android mobile device with the intent to exploit the device vulnerability and subsequently launch a malicious attack or series of attacks. The aggregation of all the attack vectors that malware writers use to infect or control Android devices to steal data and cause other damages forms the attack surface. Understanding the attack vectors helps to protect a device, system, or network against unauthorised access and consequent data breach. Mobile devices are infected when users grant permissions for the device's GPS, contacts, camera, and other resources to be accessed by malicious applications.

Sophisticated malware: Attackers spend a significant amount of effort, time, and resources developing complex variants of malware that exhibit combined characteristics in their operations. Sophisticated malware such as zero-day, metamorphic, polymorphic, among others is a set of diverse programs working collectively to accomplish attackers' intentions.

Infecting Android devices: This is when a malicious application gains access to an Android device either by authorised or unauthorised means and causes damages to the device. Other damages may include malfunctioning of legitimate applications, data extraction, theft on the device, or a combination of one or all.

1.7 Research objectives

In this thesis, we aimed to develop methods based on permission attributes to identify attack vectors and privileges that smart malware uses to attack Android OS-based devices. The techniques are focused on enabling the identification and classifications of Android malware. This included studying detection and classification techniques and tools to aid automated framework development that supports analysis of sophisticated malware. The following research objectives have been stated using Bloom's Taxonomy [43] and are subdivided into the following objective sets:

Objective Set 1:

- *Conduct an up-to-date literature review on mobile malware's current state of the art*
- *Identify malware evolution developments and infection strategies*
- *Classify evolution trends based on their sophistication and attack vectors.*

To achieve the aim of this objective, we conducted a comprehensive study on mobile evolution and infection strategies looking at different related studies previously conducted in this area. The objective aimed to advance understanding of the mobile malware history and its position in the cyber threat landscape. This is achieved by examining security threats caused on mobile devices, surveying the evolution of Android malware to identify security transformation and propagation mode adopted at each evolutionary stage, examining different evasion techniques adopted and their security implications, critically examining android malware infection strategies, raising recommendations and awareness of malware impacts and how to curb their attacks and infection rate. The extensive literature review helped in providing a solid background on malware history, invention, and detection techniques. This objective is significant because the literature review identified research gaps in the previous studies and justifies the remaining work.

Objective set output - Paper 1(Accepted): Android Mobile Malware Evolution and Infection Strategies: A survey (Chapter two).

Objective Set 2:

- *Examine existing malware detection techniques.*
- *Identify strengths and weaknesses of existing malware detection methods.*

This objective aims to examine mobile malware detection techniques to identify their strengths and limitations. To achieve this, we specified the problems to be addressed in a clear structure to identify relevant work. Multiple resources both in printed and computerised form were searched with language restriction and study selection criteria respectively. The objective was achieved through the synthesis of data comprising of tabulation of research characteristics such as detection techniques, detection approach, and accuracy, algorithms deployed, strengths and limitations respectively. The findings highlighted were then interpreted. Most of the previous works examined demonstrate their inability to detect sophisticated malware, especially those using an obfuscation approach to hide from being detected. This objective is significant because it identified the limitations and strengths in existing techniques that need improvement for better detection. Identifying some of the limitations and strengths in the existing models was leveraged on building a robust framework and techniques that detect and classify features of sophisticated and emerging mobile malware.

Objective set output - Paper 2 (Published): Analysis of Android Malware Detection Techniques: A Systematic Review (Chapter 3)

Objective Set 3:

- *Identify privileges that malware exploits to affect Android mobile devices.*

This objective aims to carry out a preliminary analysis using a small-scale malware dataset to identify privileges malware exploit to infect Android devices. To achieve this objective, we carried out a preliminary experiment to identify privilege exploitations used by malware as attack vectors. This objective is important in investigating the effectiveness of the virtual components of the malware laboratory set up, examining suitable algorithms and methods for full framework development, and identifying privileges smart mobile malware exploit to attack mobile devices. Also, techniques that were not effective during the preliminary investigation were discarded. The preliminary study helped in identifying suitable methods, techniques, and tools for full framework development.

Objective set output- Paper 3 (Published): Host-Based and Analysis of Android Malware: Implication for Privilege Exploitation (Chapter 5).

Objective Set 4:

- *Develop a framework for identifying privacy preferences such as permission features emerging and smart malware use as attack vectors on Android.*

To achieve this objective, we developed an Android malware detection framework for the identification and classification of dangerous permissions used by malware to attack Android. Android malware dataset was collected from Android malware repositories respectively which were experimented on in a controlled malware laboratory. The virtual environment was created using virtual machines. Security tools were used during setup and configurations to circumvent the escalation of malicious files and traffic to the digital forensic unit network, MOD network, and the outside world. During configuration, we adopted virtualisation methodology [44] and followed the malware lab setup principles [45] to safeguard external resources from contamination. This objective is significant because the framework provides a platform for large-scale analysis of malicious applications. The automated framework is significant in the analysis of smart and emerging malware attributes.

Objective set output- Paper 4 (Published): Android Permission Classifier (APC): A Deep Learning Algorithmic Framework Based on Protection and Threat Levels (Chapter 6)

Objective Set 5:

- *Identify the relationship between permissions requests.*
- *Identify the relationship between threat and protection level to facilitate sophisticated malware detection.*

To achieve this objective, we model techniques for visualising the correlation between dangerous permission variables that sophisticated malware deploys to affect Android mobile devices. The model identified the correlation between permission variables with flags that indicate the presence of malware based on their threat levels. This is to facilitate analysis of sophisticated malware which compromise users' confidentiality and general privacy using permissions as attack vectors.

Objective set output- Paper 5 (Accepted): Modelling Correlation between Android Permission Variables Based on Threat and Protection Level using Exploratory Factor Analysis (Chapter 7).

1.8 Research contribution to knowledge and thesis organisation

Significant contributions made in this thesis based on the research questions and objectives are summarised as follow:

1.8.1 Background and techniques

A comprehensive and systematic analysis of mobile malware evolution and infection strategies is provided in the background to form the foundation for the thesis. The first research paper (Chapter 2) presents mobile malware evolution and emerging and sophisticated malware. We carry out a comprehensive study of the mobile malware evolution and infection strategies. Based on the exhibited characteristics, we classify mobile malware evolution trends into distinct categories based on their sophistication, characteristics, and attack intents. We provide a general overview of the mobile malware infection model using infection strategies meta-data. Factors limiting the fast spread of the early mobile malware: (a) the lack of mobile OS standardization, (b) lack of mobile OS cross-platform, and (c) lack of Bluetooth technological advancements in data communication were identified. Our study identifies factors that enhance the fast spread of recent mobile malware. These include (a) the advancement in threat dimensions, (b) improved security and business communication on Tor using multi-signature transaction and encryption approach, (c) mobile platforms' integration with the IoT-based applications, (d) the emergence of evasion techniques and the advancements in mobile malware Toolkits on the dark web.

Our study provides recommendations as countermeasures to cope with the emerging mobile malware increasing threats. These include (a) the deployment of software solutions from enterprise mobility management to enhance the security of enterprise devices, (b) observation of mobile application plugins and codecs during mobile application download and installation, (c) the adoption of risk mitigation strategies by organizations, (d) development of a mobile malware infection model to understand the immunity state of a mobile device during and after recovery from malware infection.

The comprehensive analysis suggests the need to analyse comprehensively and identify strengths and limitations in the existing detection techniques. The second research paper (Chapter 3) provides a broad but thorough assessment of mobile malware detection and classification techniques. Strengths and limitations of the detection techniques were identified and carefully analysed to help develop a more robust tool for the identification of sophisticated mobile malware variants.

Chapter 4 of the thesis presents the malware lab for analysis of new generation malware. The build-up of the malware lab is composed of selected tools to aid both malicious feature extractions from Android applications and analysis. The lab architecture was built using open-source tools obtained from different reliable security platforms. The malware lab integrates both virtual machines and physical devices working together to monitor system calls and command signature features. Other components of the lab include a significant number of Android applications datasets obtained from different malicious repositories such as Genome project, ContagioDump, virus Share, and DREBIN. Part of the dataset was also collected from Aptoide online market. The lab is vital for the evaluation and automated analysis of sophisticated and new malware variants and their features.

1.8.2 Identification of attack vectors and analysis of malicious features and families

The third research paper (Chapter 5) presents how malware can be classified by statistically analysing malware with Linux malware detect (LMD) technique. LMD is a technique for mining text features by configuring functions with rules. The technique retrieves data and statistically analyse the code if obfuscated using encoding methods. We adopted the ClamAV approach [46] in the reformation of the model process for similarity classification. LMD technique extracted detected privileges exploited by sophisticated malware families. We identified permission requests such as HTTP requests, text messages, and network connection during synchronization, payloads executed, location monitoring, and battery consumption.

Understanding how malware escalates privileges is significant in combating malware because it will shift the prevention paradigm from proactive to reactive. This contribution is significant because it discloses how privileges are exploited by emerging smart malware families especially from the developmental application reuse point of view. In addition, this contribution facilitates the measurement of malware file similarities which can be used to classify malware into their

respective families automatically. It also aids malicious code module identification with their similarities during an investigation by malware experts.

1.8.3 Permission variables anomaly detection

Many frameworks recommend the use of detection engines that are based on cloud and proscribe platform-oriented approaches. Our framework adopts a different approach in providing solutions for detecting different malware variants. The fourth research paper (Chapter 6) presents different strategies for detecting smart malware based on the permission features among other features such as commands signature, API call signature, and intent features. The chapter demonstrates that detection of malicious applications based on features outsource strategy is better when compared to other strategies such as the on-platform approach. This part of the contribution is focused on the extraction of Android malicious features for examination of their consequential characteristics. Permission attributes provide hackers with the capability to exploit mobile users' information. Chapter 6 introduces a permission-based framework called Android Permission Classifier (APC) for identifying and classifying permission requests by Android malicious applications on across-the-board developments. This contribution provides malware analysts and digital forensic investigators with an efficient framework for automated detection and analysis of Android malware.

1.8.4 Bayesian model for identifying malware based on permission correlation

The fifth research paper (Chapter 7) provides two contributions associated with permission acceptance by mobile users. The key contribution is identifying the relationship that exists between two or more dangerous permission features by the analysis of their threat and protection level characteristics. The chapter proposes a protection level and permission-based taxonomy with flags that indicate the presence of dangerous permission during analysis. We identify that there are permissions with similar attributes which can trigger the installation of similar permissions with the same threat level in the background without activating further request actions for access and payloads execution. The chapter identifies that in a random sample of a given malicious dataset, some features of some variables have control or influence the characteristics of one or more variables at given levels of independent measurement.

This contribution is significant in the sense that the understanding of the correlation that exists between permission request variables aids mobile users make low-risk decisions when installing mobile applications, especially from untrusted marketplaces. The contribution also shows that it is realistic to automatically identify patterns in the similarity of permission features using threat and protection levels as parameters. This provides mobile users with an understanding of how their confidentiality, integrity, and general privacy could be compromised when credence is not paid to high-risk permission variables. This understanding will aid mobile users to prioritize their privacy over the urge to use the application especially when the application is not from trusted markets or sites.

1.9 Ethical considerations

To effectively work on malware without breaching the security and ethical rules of the University and the outside world, we acknowledged the following ethical issues and apply mitigation strategies accordingly.

1.9.1 We acknowledged that malware is dangerous

Malware is designed to replicate and infect digital devices and network infrastructures. When malware breaches the research environment, it could be dangerous to external digital devices and network infrastructures. Security procedures were adapted to contain malware and protect the wider network infrastructure in case of a security breach. Prevention procedures such as the use of security tools to monitor and analyse both the incoming and outgoing traffic bounds from the virtual machines. Network traffic was regularly checked using Wireshark to identify evidence of malware traffic. Sniffers' logs installed on the VMs were regularly checked for ICMP requests and reply traffic packets.

1.9.2 Obtaining malware to create a corpus

Many security organisations feel unsafe releasing their malware corpus for security reasons. Fear of users' intentions with the dataset and the organisations' reputation in case the data is used for crime makes it difficult for dataset release anytime malicious data set are being requested. Due to how dangerous malware is, organisations sometimes end up refusing or not willing to release the dataset for security reasons. Also, some organizations may follow bureaucratic processes which

take a long time before releasing the data. The long-time taken to obtain the data set impedes the experimental phase of the research. To gain faster access to the Android malware corpus, there are legitimate open security repositories where Android corpus was obtained. Android malware data for this research was obtained from legitimate Android malware repositories: Contagiodump [47], figshare [48], Drebin [49], among others.

1.9.3 Emulated and physical environments

As much as it is recommended to run malicious applications in an isolated environment, it is also very pertinent to note that some malware can identify the presence of virtual machines and evade detection. To avoid this, we reformed the sample name from Android-malware-samples-master.zip to train.CVS. This was to eliminate any familiarity in word detection by smart malware files in the emulated environment. Also, VMs' hypervisor was disabled to avoid VM detection by the malware according to SAN principles cited Fattori et al. [50]. The risk of physical machine contamination increases when not properly insulated from the guest operating system running VMs. They can be infected with traffics carrying malware especially when the same physical network is shared with the network of the virtual machines. To avoid this, we used sandboxing approach to add security layers to the lab during configuration. To prevent uncontrolled network communication between the VMs and the physical machines, the network of the VMs was separated from that of the physical systems by changing the network adaptor from NAT to Host-Only. Chapter 4 provided details of the lab setup.

1.9.4 Dataset storage, access, and privacy

Storage of datasets before and after the research is one of the ethical considerations for this research. When a malicious dataset is not properly stored, it can be easily accessed by unauthorized user(s) hence, escalation becomes inevitable through human error. The equipment used for the malware lab and the malware dataset were securely archived and stored in the Digital Investigation Unit (DIU) evidence store of Cranfield University for further research. The used systems were wiped to restore them to their original state. To further make the dataset secure, the stored dataset was encrypted to prevent unauthorized access. Our research, however, does not involve the use of individual personal information such as names, addresses, postcodes, health history, bank details,

etc. Therefore, personal information is not held by this study. This study has a limited risk of disclosing sensitive personal information of the researcher and other people respectively.

1.9.5 Legal issues

Computer Misuse Act [51] and the Data Protection Act 2018 [52] laws were observed to avoid the misuse of computers and other digital devices used for the research. The datasets used for the research were used solely for the research without any intent to commit a crime by infecting other devices or distributing the malicious files on the University network. Access to the malware lab and the dataset was restricted to authorised persons. The dataset was kept confidential without disclosure to unauthorised persons and made available to the authorised individuals to keep with the information security CIA triad.

REFERENCES

- [1] M. Humayun, N. Z. Jhanjhi, A. Alsayat, and V. Ponnusamy, "Internet of things and ransomware: Evolution, mitigation and prevention," *Egypt. Informatics J.*, vol. 12, no. 13, 2020, doi: 10.1016/j.eij.2020.05.003.
- [2] L. M. Cristea, "Current security threats in the national and international context," *J. Account. Manag. Inf. Syst.*, vol. 19, no. 2, pp. 351–378, 2020, doi: 10.24818/jamis.2020.02007.
- [3] T. Samme-Nlar, "The Future of Armed Conflict in Africa : What Cyber Attacks on Ethiopian Government Tells Us," *SSRN Electron. J.*, 2021, doi: 10.2139/ssrn.3705294.
- [4] N. Tariq, "Impact of Cyberattacks on Financial Institutions," *J. Internet Bank. Commer.*, vol. 23, no. 2, pp. 1–11, 2018.
- [5] W. X. Tee, W. Jung, and A. A. Radlet, "ASEAN Cyberthreat Assessment Report 2020," *Interpol*, p. 30, 2020. Accessed on: Nov. 17, 2019 [Online]. Available: https://www.interpol.int/en/content/download/14922/file/ASEAN_CyberThreatAssessment_2020.pdf.
- [6] S. MahdaviFar, A. F. Abdul Kadir, R. Fatemi, D. Alhadidi, and A. A. Ghorbani, "Dynamic Android Malware Category Classification using Semi-Supervised Deep Learning," *Proc. - IEEE 18th Int. Conf. Dependable, Auton. Secur. Comput. IEEE 18th Int. Conf. Pervasive Intell. Comput. IEEE 6th Int. Conf. Cloud Big Data Comput. IEEE 5th Cybe*, pp. 515–522, 2020, doi: 10.1109/DASC-PICom-CBDCom-CyberSciTech49142.2020.00094.
- [7] Y. Liu and C. A. Shue, "Beyond the VPN: Practical Client Identity in an Internet with Widespread IP Address Sharing," 2020 IEEE 45th Conference on Local Computer Networks (LCN), pp. 425–428, 2020, doi: 10.1109/LCN48667.2020.9314846.
- [8] C. Wheeler, "Exclusive: Relief at Old Trafford as Manchester United finally see off hackers after club was being held to RANSOM for millions of pounds," *Daily mail*, Dec. 2020. Accessed on: Dec. 04, 2020. [Online]. Available: <https://www.dailymail.co.uk/sport/sportsnews/article-9019847/Relief-Old-Trafford-Manchester-United-hackers-club-held-RANSOM.html>.
- [9] N. S. Fouad, "Securing higher education against cyberthreats : from an institutional risk to a national policy challenge," *Journal of Cyber Policy*, vol. 6, no. 2, pp. 137–154, 2021, doi: 10.1080/23738871.2021.1973526.
- [10] H. S. Lallie et al., "Cyber security in the age of COVID-19: A timeline and analysis of cyber-crime and cyber-attacks during the pandemic," *Comput. Secur.*, vol. 105, p. 102248, 2021, doi: 10.1016/j.cose.2021.102248.
- [11] Eset, "Prevent zero-day threats with powerful cloud-based sandboxing," Accessed on: Oct. 07, 2020. [Online]. Available: https://www.eset.com/fileadmin/ESET/INT/Products/Business/EDTD/v01/ESET_Dynamic_Threat_Defense_Product_Overview.pdf.
- [12] K. A. Kumar, A. Raman, C. Gupta, and R. R. Pillai, "The recent trends in malware evolution, detection and analysis for android devices," *Journal of Engineering Science and Technology Review*, vol. 13, no. 4, pp. 240–248, 2020, doi: 10.25103/jestr.134.25.
- [13] J. Wakefield, "EasyJet admits data of nine million hacked," *BBC News*, Accessed on: May. 19, 2020. [Online]. Available: <https://www.bbc.co.uk/news/technology-52722626#:~:text=EasyJet%20has%20admitted%20that%20a,debit%20card%20details%20%22accessed%22>.

- [14] J. Parsons, "Thousands of Android phones are being infected with 'Flubot' malware spreading via SMS texts," 2021. Accessed on: May. 14, 2020. [Online]. Available: <https://metro.co.uk/2021/04/29/android-phones-infected-with-flubot-malware-spreading-via-sms-texts-14491177/> (accessed Jun. 05, 2021).
- [15] R. E. P. Ort, "Weekly Threat Report 23rd April 2021," Accessed on: Dec. 22, 2021. [Online]. Available: <https://www.ncsc.gov.uk/report/weekly-threat-report-23rd-april-2021>.
- [16] NCSC, "FluBot: Guidance for 'package delivery' text message scam," Ncsc, pp. 4–7, 2021. Accessed on: Dec. 22, 2020. [Online]. Available: <https://www.ncsc.gov.uk/guidance/flubot-guidance-for-text-message-scam>.
- [17] C. Currie, "Impact of mobile computing: anticipating the effects on the campus," *Plan. High. Educ.*, vol. 41, no. 2, pp. 169, Accessed on: Dec. 22, 2020. [Online]. Available: <https://link.gale.com/apps/doc/A342177551/AONE?u=anon~bcea224b&sid=googleScholar&xid=451f18a9>.
- [18] I. Davydova, L. Didenko, and V. Tomina, "Legal Nature And Inheritance Of Virtual Property In U Kraine And The World : Current," *law journal*, vol. 10, no. 2, pp. 1–26, 2021. doi: <https://doi.org/10.31207/ih.v11i2.283>
- [19] S. G. Stats, "Browser Market Share Worldwide," Accessed on: Dec. 22, 2020. [Online]. Available: <https://gs.statcounter.com/os-market-share/mobile/worldwide>.
- [20] A. Laughlin, "More than one billion Android devices at risk of malware threats Read more: <https://www.which.co.uk/news/2020/03/more-than-one-billion-android-devices-at-risk-of-malware-threats/> - Which?," *Uk*, Mar. 06, 2020.
- [21] Y. Yan et al., "Understanding and Detecting Overlaybased Android Malware at Market Scales. In *The 17th Annual International Conference on Mobile Systems, Applications, and Services*, June 17–21, 2019, Seoul, Republic of Korea. ACM, New York, NY, USA. doi:<https://doi.org/10.1145/3307334.3326094>
- [22] K. Chinetha, J. D. Joann, and A. Shalini, "An Evolution of Android Operating System and Its Version," *Int. J. Eng. Appl. Sci.*, vol. 2, no. 2, pp. 30–33, 2015. Accessed on: Dec. 22, 2020. [Online]. Available: <https://media.neliti.com/media/publications/257997-an-evolution-of-android-operating-system-2d35484a.pdf>.
- [23] C. Wang, "The research of Android System Architecture and application programming," *Proceedings of 2011 International Conference on Computer Science and Network Technology*, Harbin, China, 24–26 Dec. 2011, doi: 10.1109/ICCSNT.2011.6182081
- [24] Z. Ren, H. Wu, Q. Ning, I. Hussain, and B. Chen, "Ad Hoc Networks End-to-end malware detection for android IoT devices using deep learning," *Ad Hoc Networks*, vol. 101, p. 102098, 2020, doi: 10.1016/j.adhoc.2020.102098.
- [25] J. L. John, "Forensics and Preservation," Accessed on: Dec. 22, 2020. [Online]. Available: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.384.6486&rep=rep1&type=pdf>.
- [26] S. Dogan and E. Akbal, "Analysis of Mobile Phones in Digital Forensics," 2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), Opatija, Croatia, 22–26 May, 2017, doi: 10.23919/MIPRO.2017.7973613.

- [27] S. Li, T. Qin and G. Min, "Blockchain-Based Digital Forensics Investigation Framework in the Internet of Things and Social Systems." In *IEEE Transactions on Computational Social Systems*, vol. 6, no. 6, pp. 1433-1441, 2019, doi: 10.1109/TCSS.2019.2927431.
- [28] Y. K. Gupta, "Systematic Digital Forensic Investigation Model," *International Journal of Computer Science and Security (IJCSS)*, vol. 5, no. 1, pp.118-131, 2016.
- [29] S. V. Korolev and S. I. Kuchanov, "Calculation of the distribution of units and composition heterogeneity of copolymers obtained by a complex-radical mechanism," *Polym. Sci. U.S.S.R.*, vol. 24, no. 3, pp. 709–719, 1982, doi: 10.1016/0032-3950(82)90064-8.
- [30] N. Idika and A. Mathur, "A survey of malware detection techniques," Accessed on: Dec. 22, 2020. [Online]. Available: <https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.75.4594&rep=rep1&type=pdf>
- [31] C. C. Joldes, "Proceedings of the 14 th International Conference Accounting and Management Information Systems AMIS IAAER 2019 Bucharest University of Economic Studies," 2020.
- [32] W. Han, J. Xue, Y. Wang, F. Zhang, and X. Gao, "APTMallInsight : Identify and cognize APT malware based on system call information and ontology knowledge framework," *Inf. Sci. (Ny)*., vol. 546, pp. 633–664, 2021, doi: 10.1016/j.ins.2020.08.095.
- [33] S. Alsoghyer and I. Almomani, "Ransomware Detection System for Android Applications," *Electronics*, vol. 8, no. 8, pp. 868, Aug. 2019. doi:<https://doi.org/10.3390/electronics8080868>.
- [34] S. Chuprat et al., "Malware Forensic Analytics Framework Using Big Data Platform," *Proceedings of the Future Technologies Conference (FTC) 2018*. FTC 2018. *Advances in Intelligent Systems and Computing*, vol. 881, Springer, Cham. https://doi.org/10.1007/978-3-030-02683-7_19.
- [35] R. Fedler, M. Kulicke, and J. Schutte, "An antivirus API for Android malware recognition," *Proc. 2013 8th Int. Conf. Malicious Unwanted Softw, "The Am. MALWARE Fajardo*, PR, USA, 22-24 Oct. 2013, pp. 77–84, 2013, doi: 10.1109/MALWARE.2013.6703688.
- [36] Y. Pan, X. Ge, C. Fang, and Y. Fan, "A Systematic Literature Review of Android Malware Detection Using Static Analysis," *IEEE Access*, vol. 8, pp. 116363–116379, 2020, doi: 10.1109/ACCESS.2020.3002842.
- [37] V. Kouliaridis, G. Kambourakis, D. Geneiatakis, and N. Potha, "Two anatomists are better than one-Dual-level android malware detection," *Symmetry (Basel)*., vol. 12, no. 7, pp. 1–21, 2020, doi: 10.3390/sym12071128.
- [38] V. Syrris and D. Geneiatakis, "On machine learning effectiveness for malware detection in Android OS using static analysis data," *J. Inf. Secur. Appl.*, vol. 59, no. May, p. 102794, 2021, doi: 10.1016/j.jisa.2021.102794.
- [39] A. De Lorenzo, F. Martinelli, E. Medvet, F. Mercaldo, and A. Santone, "Visualizing the outcome of dynamic analysis of Android malware with VizMal," *J. Inf. Secur. Appl.*, vol. 50, p. 102423, 2020, doi: 10.1016/j.jisa.2019.102423.
- [40] H. Hasan, B. Tork Ladani, and B. Zamani, "MEGDroid: A model-driven event generation framework for dynamic android malware analysis," *Inf. Softw. Technol.*, vol. 135, no. 6, 2021, doi: 10.1016/j.infsof.2021.106569.
- [41] A. Afianian, S. Niksefat, B. Sadeghiyan, and D. Baptiste, "Malware Dynamic Analysis Evasion Techniques: A Survey", *ACM Comput. Surv.*, vol. 52, no. 6, pp 28, 2019. doi: <https://doi.org/10.1145/3365001>.

- [42] Z. Akhtar, "Malware Detection and Analysis: Challenges and Research Opportunities," Accessed on: Dec. 22, 2020. [Online]. Available: <http://arxiv.org/abs/2101.08429>.
- [43] I. M. Arieviditch, "The vision of Developmental Teaching and Learning and Bloom's Taxonomy of educational objectives," *Learn. Cult. Soc. Interact.*, vol. 25, p. 100274, 2020, doi: 10.1016/j.lcsi.2019.01.007.
- [44] L. K. Yan, M. Jayachandra, M. Zhang, and H. Yin, "V2E: Combining hardware virtualization and software emulation for transparent and extensible malware analysis," *ACM SIGPLAN Not.*, vol. 47, no. 7, pp. 227–237, 2012, doi: 10.1145/2365864.2151053.
- [45] A. Mills and P. Legg, "Investigating Anti-Evasion Malware Triggers Using Automated Sandbox Reconfiguration Techniques," pp. 19–39, 2020, doi: 10.3390/jcp1010003.
- [46] N. L. Or, X. Wang, and D. Pao, "MEMORY-Based Hardware Architectures to Detect ClamAV Virus Signatures with Restricted Regular Expression Features," vol. 65, no. 4, pp. 1225–1238, 2016. doi: 10.1109/TC.2015.2439274.
- [47] E. Yu. "Contagiodump project" Android MalwareSamples, Accessed on: Dec. 22, 2020. [Online]. Available: <https://github.com/ashishb/android-malware/find/master>.
- [48] Yerima S., "Android malware dataset for machine learning", Accessed on: Dec. 22, 2020. [Online]. Available: https://figshare.com/articles/Android_malware_dataset_for_machine_learning_2/5854653.
- [49] The Drebin Dataset, Accessed on: Dec. 22, 2020. [Online]. Available: The Drebin Dataset (tu-bs.de).
- [50] A. Fattori, A. Lanzi, D. Balzarotti, and E. Kirda, "Hypervisor-based malware protection with AccessMiner," *Comput. Secur.*, vol. 52, pp. 33–50, 2015, doi: 10.1016/j.cose.2015.03.007.
- [51] ONS, "Overview of fraud and computer misuse statistics for England and Wales," Accessed on: Dec. 22, 2018. [Online]. Available: <https://www.ons.gov.uk/peoplepopulationandcommunity/crimeandjustice/articles/overviewoffraudandcomputer misusestatisticsforenglandandwales/2018-01-25>.
- [52] R. G. Priest, "Data Protection Act," *Bull. R. Coll. Psychiatr.*, vol. 12, no. 5, pp. 204–204, 1988, doi: 10.1192/s0140078900020162.

CHAPTER 2: ANDROID MOBILE MALWARE EVOLUTION AND INFECTION STRATEGIES: A SURVEY

The content of this chapter is published in the International Journal of Information Security and Cybercrimes Research with authors Moses Ashawa and Morris Sarah.

Citation: M. Ashawa and S. Morris, “Analysis of Mobile Malware: A Systematic Review of Evolution and Infection Strategies,” vol. 4, no. 2, pp. 1–29, 2021, doi: 10.26735/KRVI8434.

This chapter is linked to the first objective of the research. Recent cases of malware attacks and their impacts were highlighted in chapter 1. This motivated us in surveying the existing mobile malware attack cases. Cases of malware attacks and economic impacts as discussed in **Sections 1.2 and 1.1.2** keep increasing. This chapter investigated mobile malware evolution state of the art. During the survey, we identified some mobile malware infection strategies and propagation modes to help reduce the infection rate. Based on objective set 1, we carried out an extensive survey of mobile malware evolution and state of art by examining famous incidents of mobile malware attacks. We focused on mobile OS platforms attacked, the name of the malware, the type of mobile malware, and their mode of propagation.

In addition, we identified factors that enhance the fast spread of recent mobile malware such as the advancement in threat dimensions, improved security and business communication on Tor using multi-signature transaction and encryption approach, mobile platforms’ integration with the IoT-based applications, the emergence, and advancement in evasion techniques, and the advancements in mobile malware Toolkits on the dark web, among others. The identification of these factors was leveraged to carry out a systematic study in **chapter 3 (published in paper 2)** to identify the strengths and limitations of the existing detection techniques.

Abstract

The open-source and popularity of Android attracts hackers and has multiplied security concerns targeting devices. As such, malware attacks on Android are one of the security challenges facing society. This paper presents an analysis of mobile malware evolution between 2000-2020. The paper presents mobile malware types and in-depth infection strategies malware deploys to infect mobile devices. Accordingly, factors that restricted the fast spread of early malware and those that enhance the fast propagation of recent malware are identified. Moreover, the paper discusses and classifies mobile malware based on privilege escalation and attack goals. Based on the reviewed survey papers, our research presents recommendations in the form of measures to cope with emerging security threats posed by malware and thus decrease threats and malware infection rates. Finally, we identify the need for a critical analysis of mobile malware frameworks to identify their weaknesses and strengths to develop a more robust, accurate, and scalable tool from an Android detection standpoint. The survey results facilitate the understanding of mobile malware evolution and the infection trend. They also help mobile malware analysts to understand the current evasion techniques mobile malware deploys.

Keywords: Infection strategies, malware evolution, mobile malware, OS target, cyber threat

2.1 Introduction

The world is undergoing a rapid information transformation piloted by redefining and restructuring technological processes. This fast technological growth has caused the evolution and advancement of major security issues worldwide, in addition to its many benefits (e.g., fast financial transactions, automation, short-time processing of data) [1-2]. Information technology usage ushers in new security challenges (e.g., impersonations, sensitive data exposure, malware attacks, other cyber-threat issues) [3]. Malware has become a global issue and is now a sophisticated threat that information security is battling on a national and international scale. Cyber-threats have increased significantly during the COVID-19 pandemic as cybercriminals have taken advantage of the COVID outbreak, especially during the adaption of the work from home (WFH) approach to curbing the spread of the virus [4-5].

While the pandemic claims human lives in the millions and creates anxiety and considerable uncertainty, cyber-attacks are also harming peoples' livelihoods. Government and non-governmental organizations have been compelled to adopt precautionary measures and adapt to remote working during the pandemic, with little or no adequate and necessary training of their employees to be cyber smart when working remotely. Even for those that have the training, working from home on a poorly secured network is a concern. As a result, cyber-criminals are exploiting individuals and government vulnerabilities for financial gain through pandemic-related threats such as fraud, phishing, and malware. Among the cyber threats, the research of Cristea [6] reported that malware attacks on mobile devices are the most challenging current security threat in the national and international context. Malware has evolved and has become sophisticated, so detecting some variants becomes very difficult. McLaren et al. [7] stated that bot writers deploy evasion techniques like code encryption to make detection by pattern recognition less effective.

The advancement in mobile technology has brought about corresponding effects on mobile malware, which attacks mobile devices. Among the mobile OS, Android is the fastest-growing OS used worldwide [8-10]. Research has shown that the Android operating system outran other mobile OS due to some factors such as ease of use, affordability, open-source code, and compatibility commitment [11-13]. The first malware, called Creeper [14], was created to target personal computers, but as technology advanced, malware evolved to attack mobile devices as well. The first mobile malware, known as Cabir [15], was designed to target Symbian-based mobile platforms. Since the advent of Cabir, mobile malware has evolved into different variants and with differing complexities. Unlike PC malware, mobile malware has become so pronounced due to the enormous amount of personal and financial information cybercriminals harvest from those devices. Consequently, data breaches and financial loss cases caused by mobile malware keep increasing.

The research published by the UK Department for Digital, Culture, Media & Sport in 2020 shows that malware caused 87% of the total cybersecurity attacks that caused financial loss [16]. In addition, the growth of malware attacks on mobile devices continues to increase despite the number of industrial solutions, scant systematic reviews, and meta-analyses focusing on identifying strategies mobile malware deploys as infection strategies or attack vectors to infect mobile devices. Also, while there are several mobile malware detections, classification, and analysis techniques, the systematic review and meta-analysis of the infection strategies are deficient. Our systematic meta-analysis aims to look at the mobile malware evolution and infection strategies. Many mobile

malware attack goals can be fragmented into the motives and the behaviour related to the motive. The World Economic Forum (WEF) threat risk report highlighted that every cyberattack caused by malware leaves indelible impacts, even after the malware has been detected and removed [17].

This paper has the following aims: (1) To advance understanding of the mobile malware history and its position in the cyber threat landscape, examining the major security threats to mobile devices; (2) To provide a discussion on the evolution of Android malware to identify security transformation and propagation modes adopted at each evolutionary stage; (3) To examine different evasion techniques adopted and their security implications; and (4) To critically examine android malware infection strategies and raise recommendations and awareness of malware impacts and how to curb the attacks and infection rates. Our research made the following contributions:

1. Providing an up-to-date study of mobile malware evolution and infection strategies.
2. Classifying mobile malware evolution trends into distinct categories based on their sophistication, characteristics, and the motivations behind attacks.
3. Providing a general overview of the mobile malware infection model based on the meta-analysis of the infection strategies.
4. Identifying factors that limited the fast spread of the early mobile malware. These include (a) the lack of mobile OS standardization, (b) the lack of mobile OS cross-platform, and (c) the lack of Bluetooth technological advancements in data communication.
5. Identifying factors that enhance the fast spread of recent mobile malware. These include (a) the advancement in threat dimensions, (b) improved security and business communication on Tor using a multi-signature transaction and encryption approach, (c) mobile platforms' integration with the IoT-based applications, and (d) the emergence of evasion techniques and the advancements in mobile malware Toolkits on the dark web.
6. Providing recommendations as countermeasures to cope with emerging mobile malware and its increasing threats. These include (a) the deployment of software solutions from enterprise mobility management to enhance the security of enterprise devices, (b) observation of mobile application plugins and codecs during mobile application download and installation, (c) the adoption of risk mitigation strategies by organizations, and (d) development of a mobile malware infection model to understand the immunity state of a mobile device during and after recovery from malware infection.

The remainder of the paper is structured as follows: Section II provides the research methodology, section III provides the background of the study, Section IV presents the discussion, Section V presents the infection and attack vectors, and Section VI presents the conclusion.

2.2 Methodology

Scientometrics was used to assess and analyse the finite features of the selected papers used for this study [18]. The source of literature for this study came from the Web of Science collections. Field tags such as parenthesis, wildcards, Boolean OR operator, and quotation marks were used to create the query for exact words and phrases. Two datasets (core dataset (CD) and extended dataset (ED)) were ultimately used for our study. Web of Science retrieval was used to obtain the core dataset which contained many kinds of literature. For further filtering, the repeated literature was eliminated using CiteSpace, a function of the Web of Science (as shown in Figure 2-1). The final database was then generated, combining the core and expanded dataset containing the reviews and the articles. The topic search (TS) was used as the target to perform a holdout data search to pull out all the keywords essential for both the core and extended dataset. This was to determine the total increase in node purity of the most important variables (Keywords) in the TS (figure 2-2). In summary, different articles were collected during the filtering process. However, only 243 papers met our selection criteria, including the period and language. Some papers contained keywords, but the content was written in languages other than English. Many more were eliminated according to our selection criteria.

To discover the essential organization of the CD and ED, we conducted an exploratory factor analysis [66] using the applied rotation method, simplimax, to identify latent keywords that were not observed but share common variances. The aim of conducting the exploratory factor analysis on the combined dataset generated was to enhance the complexity reduction of the large data (articles and reviews). The exploratory factor analysis brought out the factor correlations and the uniqueness characterising the CD and ED articles with their influencing TS tag-keywords (as shown in figure 3-3).

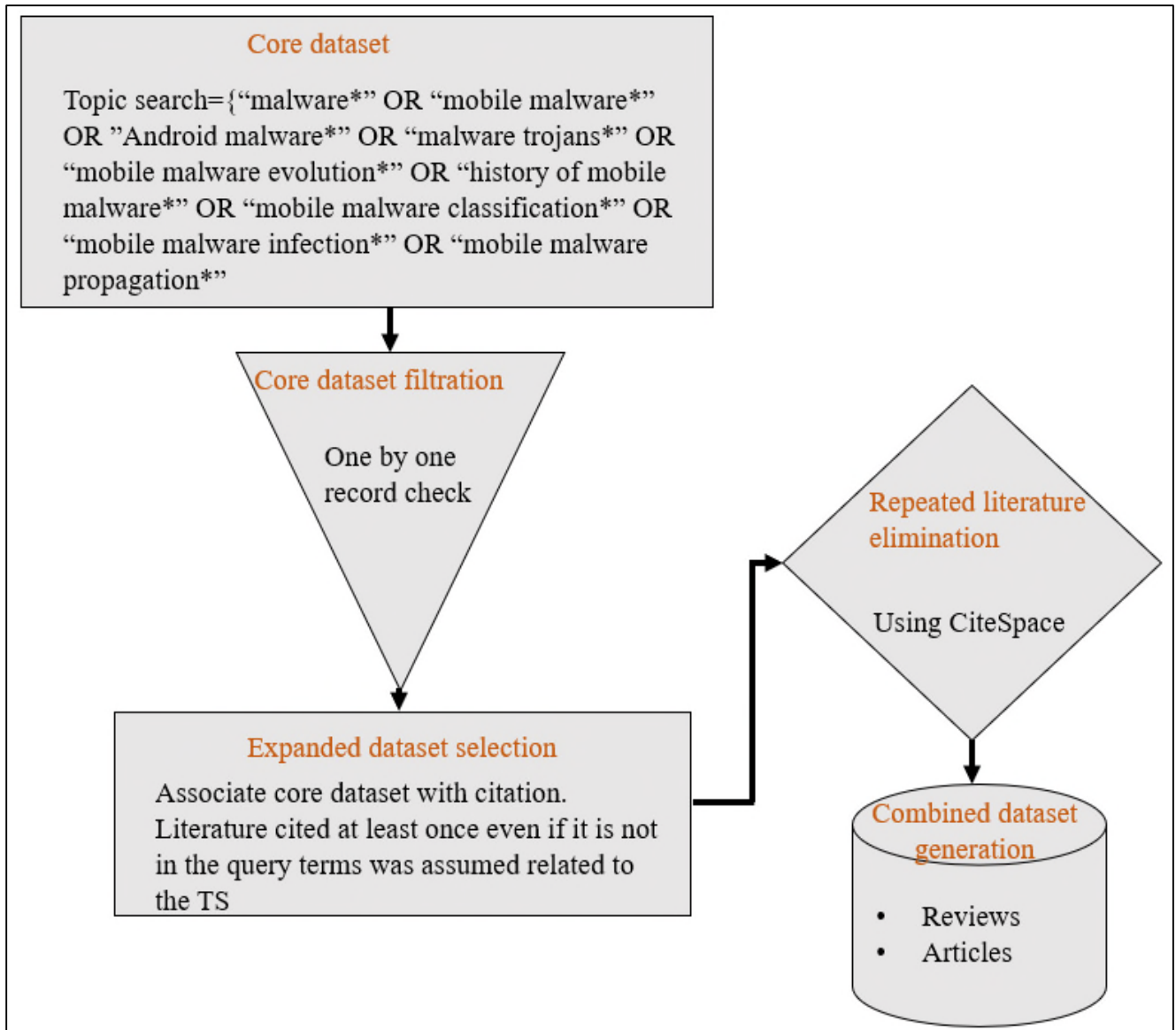


Figure 2-1 Scientometrics dataset selection, filtration, and elimination process

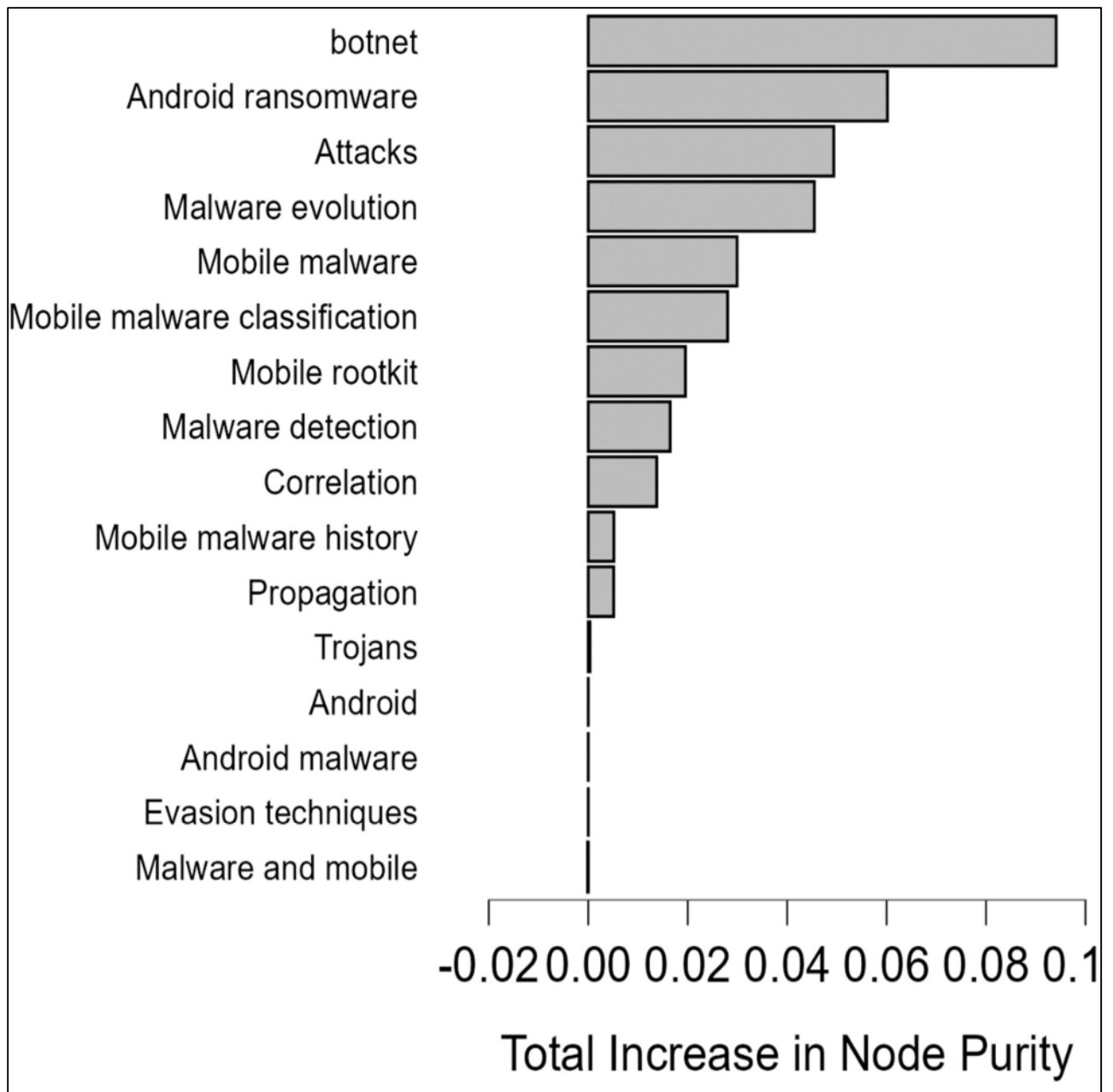


Figure 2-2 Determining the total increase in node purity of the most important keywords in the Topic Search from the combined dataset

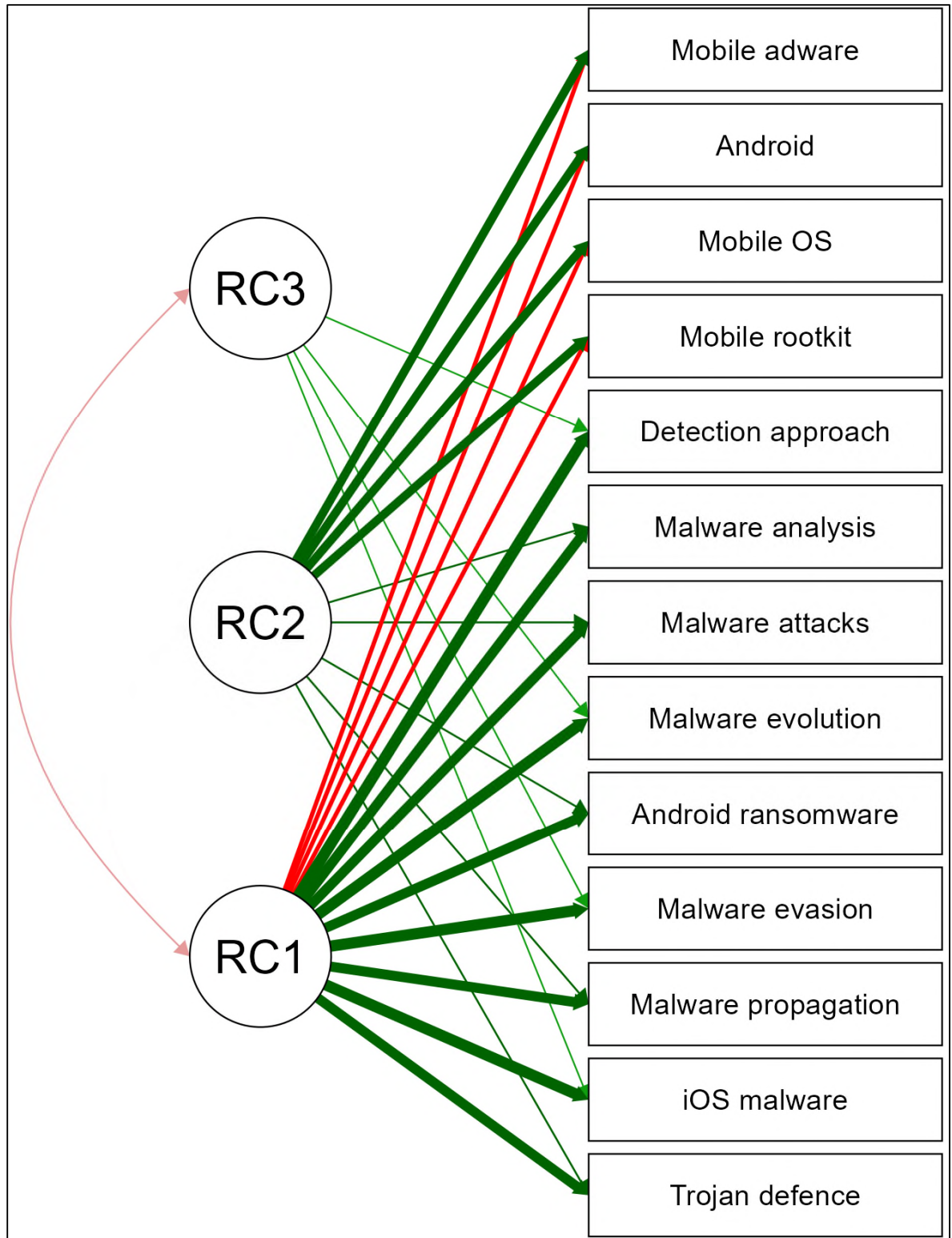


Figure 2-3 Path diagram showing the keywords characterization and the number of factors using simplimax as the rotation method and minimum residual as the estimation method

2.3 Background

2.3.1 History of mobile malware

General malware appeared in the 1970s when Bob Thomas wrote a self-replicating virus called Creeper Worm. Creeper worm was Adware that displayed popups on systems with the message “I’m the Creeper, catch me if you can!” [19-21]. After the emergence of Creeper, other significant PC malware such as Wabbit [22], Elk Cloner [23], Brain Boot Sector Virus [24], PC-Write Trojan [25], Morris Worm [26], Michelangelo Virus [27], and Melissa Virus [28] also emerged. The emerging technology in mobile devices has brought about a proportional malware threat to mobile devices. Smartphones are speedily substituting PCs both in the workplace and at home. Most activities such as web surfing and financial transactions rely on mobile devices. Due to the increased dependence on smartphone devices and the financial information they store, malware writers’ attention has significantly shifted from PC to mobile devices.

According to Shah et al. [29], the history of malware attacks on mobile devices can be traced back to 2004 when the first mobile malware called Cabir was written by Vallez to target Symbian-based devices. Being a worm, Cabir injected its payload into the victim’s devices via Bluetooth file-sharing as a spreading mechanism. The malware was not sophisticated in its operation. It displayed annoying popups that made the infected mobile device unpleasant to operate. Due to the limitations of Blue-tooth technology in data communication, slow data speed, and short signals, the rapid spread of the malware was impeded. Another factor that impeded the rapid spread of Cabir was the fact that the devices had to be close to each other to establish a connection before the infection could occur. The lack of standardization in mobile OS was also another limiting factor that inhibited the rapid spread of Cabir.

Another mobile malware called Skulls [30] was developed in the same year with criminal motives and later formed the basis of PC malware. The later malware was found to overwrite mobile application files and substituted applications’ icons with crossbones and skulls that stopped mobile device functionalities. In 2005, malware moved into the realm of information theft when more dangerous mobile malware called Commwarrior and Pbstealer [31] was discovered targeting the Nokia 7160. Both Commwarrior and Pbstealer targeted and harvested information-sensitive data such as passwords and usernames using nearby connected mobile devices. Looking at the malware

pattern, the study by Mayrhofer et al [32] shows the paradigm shift in malware development from Symbian devices to Android, particularly the development of malware for Java 2 Micro Edition (J2ME). The major trend followed by malware writers in this paradigm was developing malicious codes using J2ME to send premium-rate texts or using SMS as a social engineering mechanism to trick victims into confirming non-existing financial operations.

The first android mobile malware was called “AN-DROIDOS_DROIDSMS.A” [33]. It was detected as a Trojan in 2010 by Trend Micro, while the Ikee worm was also uncovered for iOS-based devices (Apple iPhones). The mobile infection trend in this era focused on phone jailbreaking, rooting, Rick-rolling, and changing the phone background to display Rick Astley’s image leaving the message on the screen “Ikee is never gonna give you up” [34]. Android malware was originally spread by using third-party application marketplaces, due to android openness in its App ecosystems circulation, which undoubtedly accelerated this misuse, unlike Apple.

From 2011 till date, there has been a progressive growth of malware threat complexity. For Android, the kill switch was used by Google, but the challenge remains that the kill switch will not be effective if an earlier download has an infection on the device. iPhone/FindAndCall [35] and Android/ DroidKungFu [36] malware snip individual data and forward this personal information to remote network servers. Other malicious software such as FinSpy, Android/Nickispy, and Android/Spybubble has been developed to record mobile phone calls, send SMS, and monitor mobile location as spy instruments. The research of [37] observed that the number of malware attacks on android from 2013 grew to 98.05% compared to Symbian (0.27%), J2ME (1.55%), and others (0.13%) (as shown in figure 2-4).

The study by Chen et al [38] asserted that malware infections and threats on mobile devices are increasingly spreading to affect other smart devices such as smart cities, smart TVs, smartwatches, cloud-based technology, and IoT device platforms. This shows the susceptibility of smart devices to attacks such as buffer overflow, which can be deployed by hackers to remotely control those devices. Apart from attacks on smart devices, the medical field has recently been severely affected by the number of malware attacks. Malware attacks cardiac devices and reconfigures them [39], resulting in failures during cardiovascular implantation and, thus, putting the lives of patients at risk. The attack of WannaCry Ransomware [40] on the UK NHS infrastructure in 2017 affected several NHS computers have a considerable effect on the system operation and patient records.

From the above history, we noted that many malware codes were written for fun and probably for behavioural testing of software from the beginning. However, evolution in writing malware has increased the complexity of its impacts on financial institutions, businesses, and information leakage. It has evolved to be used for financial gain, vengeance, system sabotage, cyberstalking, and political influence. Table 2-I shows a detailed review of major mobile malware incidents and evolution and propagation strategies.

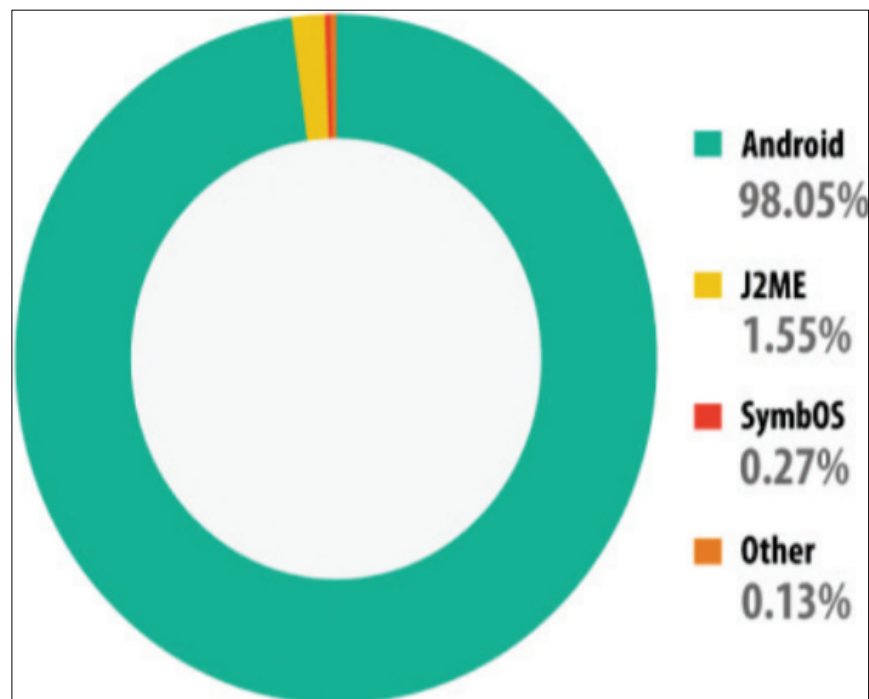


Figure 2-4 Comparison of malware attack rates on mobile OS platforms in 2013 [38]

Table 2-1 Mobile malware incidents and evolution

Year	Research	OS Targeted	Malware	Malware type	Malware functionality
2000	[90][91][92]	Symbian	Timofonica	Worm	Propagates through Bluetooth
2004	[93][94][95]	Symbian	Cabir	Worm	Propagates through Bluetooth
2004	[96][97][98]	Symbian	Skull	Trojan	Displays icons and replaces system applications and files
2004	[99][100][101]	Windows CE	Brador	Backdoor	Gains remote access to a network
2005	[102][103][104]	Symbian	Commwarrior	Worm	Infects files by propagating through MMS and Bluetooth
2005	[105][106]	Symbian	Locknut	Trojan	Enables installation of corrupted applications
2005	[97][106][107]	Symbian	Drever	Trojan	Swaps antivirus loaders
2005	[108][109]	Symbian	Skudoo	Trojan	Installs Skull and Cabir
2005	[110]	Symbian	Singlejump	Trojan	Disables system functions
2005	[111]	Symbian	Cardtrap	Trojan	Deletes antivirus files
2005	[62][122]	Symbian	Pbstealer	Trojan	Steals sensitive data
2006	[113]	J2ME	RedBrowser	Trojan	Sends premium SMS
2006	[114]	Symbian	Rommwar	Trojan	Replaces device applications
2007	[115]	Symbian	Flexispy	Trojan	Steals sensitive data
2009	[35]	iOS	Ikee	Worm	Jailbreaks Apple devices
2010	[35]	Android	DroidSMS.A	Trojan	Sends premium SMS
2010	[116]	Blackberry, Android, windows, and Symbian	Zitmo (Zeus-in-the-mobile)	Trojan	Online banking attacks
2010	[117]	Android	Tap Snake	Spyware	Location monitoring
2010	[45][118]	Android	FakePlayer	Trojan	Sends premium SMS
2011	[119]	Android	DroidDream	Trojan	Roots devices, installs other malicious apps at the backdoor

2012	[120]	Android	Boxer	Trojan	Sends premium SMS
2012	[121][122]	Android	Opfake	Trojan	Device rooting
2012	[122]	Android	Fakeinst	Trojan	Performs update attacks
2013	[123][124]	Android	fakeDefender	Ransomware	Prompts users to buy security app
2013	[125]	Android	Obad	Backdoor	Zero-day attack
2014	[68][126]	Android	NotCompatible.CA	Trojan	Side loading apps to hinder security assurance.
2015	[127]	Android	Acecard	Trojan	Banking Trojan
2015	[128]	iOS	XcodeGhost	Trojan	Overlays apps, Steals, and uploads user data to C2 servers
2015	[98]	iOS	YiSpecter	Adware	Attacks both jailbroken and non-jailbroken iOS phones
2016	[92]	Android, iOS	HummingBad	Trojan	Rogue software app
2016	[129]	Android	Xbot	Ransomware	Sends premium SMS and steals banking details
2016	[130]	Android	AndroidOS.Fusob	Ransomware	Remotely access infected devices. demands ransom
2016	[131]	iOS	AceDeciever	Trojan	Exploits Apple DRM design flaws
2017	[6][132]	Android	Ztorg	Trojan	Roots devices
2017	[133]	Android	ToastAmigo	Backdoor	Deploys toast overlay attack to install more malicious apps
2018	[80][134]	Android	Chamois	Backdoor	steals OAuth tokens
2018	[135]	iOS	Pegasus	Spyware	Records calls, keylogging, zero-day exploit
2019	[136][137]	Android	TimpDoor	Spyware	Click fraud attack
2019	[11][138]	Android	Cerberus	Trojan	Intercepts calls
2019	[12][139]	Android	XHelper	Trojan	Displays popup adds, redirects users
2020	[140]	Android	Ghimob	Trojan	Demands ransom

2.3.2 Evolution of Android malware

Android open model architecture is one of the potential factors that encouraged fast malware progression, where the Google bouncer uploads malware applications written by hackers on the play store without adequate security checks. The Android open framework model is one of the major factors that pose a security risk. The open-source nature of the Android platform enables OS modification by the manufacturers for feature enhancement and thus makes the source code susceptible to attackers. It also provides opportunities for tinkering the OS-based devices, thus, weakening the security of the device.

In 2010, AndroidOS.DroidSMS.A [41] emerged as the first Android mobile malware. AndroidOS.DroidSMS.A is a fraudulent application for the SMS premium rate. Using SMS services, the Trojan subscribes to victims' Android devices. The affected devices received different text messages at a premium rate using automatic subscription premium SMS service. AndroidOS.DroidSMS.A installs itself on the phone once permission request attributes related to SMS features were granted. Another Trojan discovered in the same year was Tap Snake [42], which propagated via HTTP by acquiring the victim's device location via GPS or network services. Tap Snake masqueraded as a mobile game but was a spy tool, remotely monitoring mobile conversations and locations. The monitored data are then forwarded to malicious servers in the background for further attacks and vulnerability exploits. AndroidOS.DroidSMS.A and Tap Snake Trojans were wreaking havoc on Android in 2010. Their counterpart Ikee worm [43-44] was jailbreaking iOS-based mobile devices to use an SSH password. Based on the meta-analysis of their operation mode, we can assert that it was very easy to avoid being infected by Tap Snake if users pay attention to the services the application requests to access. For instance, during the installation, it requests permission to access network communication, device location, and system tools. We can infer that this infection could have been avoided if users prioritise their security over the need to use the masquerading applications.

Three other Android malware that emerged in 2010 included SMSReplicator, Geinimi, and Fakeplayer [45]. Both AndroidOS.DroidSMS.A and the Tap Snake game did not infect many devices, because the attack vectors were limited due to a lack of cross-platform propagation ability. According to the research of [46-47], another factor that limited the spread of those Trojans was that Android (4.45%) was still not as popular as Symbian OS (34.33%), iOS (32.92%), and BlackBerry OS (10.16%). Apart from cross-platform propagation limitation and lack of Android popularity, other factors such as low mobile features like weak internet connectivity and short Bluetooth range

were also contributing factors that repressed rapid propagation of the first emergent malware. The common behavioural goal amongst the two malware applications is their monetization motive towards the mobile space.

While only a few Android malware families emerged in 2010, the number of Android malicious applications multiplied greatly in 2011 with over 44 different Android malware families discovered. Among the variants discovered, DroidDream [48] was one of the notable families that rooted the victims' devices to steal sensitive data. Thus, android malware evolved from SMS premium to modification of mobile functionalities. While the family of DroidDream focuses on stealing users' private information, the Genimi family [49] takes control from the attacker's remote server. DroidDream was an Android botnet type of malware that stole Unique Identification Information (UII) by gaining root access to the victim's Android device. Hackers were able to control the infected devices remotely by abetting automatic download of other malicious files without the victim's awareness or authorisation.

As noted in the study of Yan and Yan [50], the emergence of this malware family led to the mainstream Android malware Toolkits. This became the foundation of the mobile cybercrime market worldwide. As the attack success progressed, Droid-Dream and its likes were sold on the dark web illegally where virtual access can be deployed to harm. This evolution made Android-malware-spreading kits available and accessible worldwide. We can infer that this evolution made buying, ownership, and deployment of Android malware easy. Hackers pay for the toolkits in Bitcoin and are provided with the necessary tutorials needed for effective Android malware infection. This led to a rapid proliferation of malware samples and the corresponding attacks.

Opfake [51] and Fakeinst [52] were the most prominent Android Trojans discovered by Kaspersky in 2012, with over 1,083 Android malware samples discovered by different security companies such as Kaspersky and F-Secure. Most of the Android malware samples were the repackaged versions of legitimate applications, which lead to the policing requisite of developing techniques that could detect repackage applications. Our research observed that this malware family adopted drive-by downloads and updated attacks to infect victims' devices, which was more difficult to detect. According to Meng [53], samples of the families discovered in 2012 could influence root exploits to completely compromise the security of Android. Thus, this presented a high level of privacy threats to the Android community. Some of the Android malware families in 2012 had inherent attributes of the 2010 and 2011 samples.

Similar attributes included sending premium text messages, turning compromised devices into a botnet, harvesting users' account details and calling in the background without the awareness of the owner. Root-level exploitation by these Android malware families caused poor security stability of the Android OS, leading to other different consequences such as device bricking, loss of device warranty, financial loss, and more openings of attack surfaces to make the compromised devices more vulnerable to attacks. Other Android malware, Plankton [54-56] and Foncy IRC bot later emerged towards the end of 2012. Kaspersky reported that the latter was not as sophisticated and prevalent as Opfake and Fakeinst.

While mobile security was devising strategies to tackle the spread of emerged Android malware, new variants of Android malware emerged in 2013, namely Fave-Av-Reader, Plapka, Simplocker, Rough_Skype, Dendroid [57], among others. The most sophisticated among thousands of variants discovered in 2013 was Obad. Android malware Code obfuscation was the first experience with the emergence of Obad Trojan. Writers of Obad infused obfuscation techniques that enabled Obad to take the role of device administrator in the background. We can infer that Obad was a backdoor Trojan that used cracked sites and Playstore for pinging precise resources through pre-defined SMS strings. Financial platforms such as mobile banking apps, e-wallets, and credit card thefts were major attack goals for the Trojan, as highlighted by Austin [58]. The emergence of this malware trend leads to an increase in Android cybersecurity black markets where stolen data are sold. The obfuscation method adapted by 2013 Android malware developers to evade detection by anti-malware engines is a factor that advances the fast propagation of Obad Trojan. Due to the obfuscation technique, the static analysis would be much harder to achieve by cybersecurity experts and anti-virus engineers than could be done on the previous trends. While root level exploit was common in the previous trend (2012), Obad exploits zero-day weaknesses to gain more and higher privileges on the infected devices.

The 4G-enabled Android devices involved a technological shift that motivated malware writers to exploit further vulnerabilities existing on Android phones via off-the-shelf creation of malware tools and deployment strategies. This led to many Android malware families being discovered in 2014; many samples displayed the same features seen in the previous variants. The notable Android malware family that exhibited unique characteristics from the already known samples between 2010 and 2013 is NotCompatible.CA as highlighted by Ariyapala et al. [59]. NotCompatible.C is self-protecting and persistent through encryption and redundant activities, thus making static analysis

hard to achieve. Being a botnet kind of Android malware, over 8.5 million devices were infected via spam emails.

Notable impacts of NotCompatible.C included WordPress accounts cracking and the bulk purchase of event tickets from the infected devices. Unlike previous families, dynamic analysis became very difficult on NotCompatible.C due to its awareness of the emulated and virtual environments. Due to its obfuscated method, it can evade the vetting processes on play store and anti-virus products. Scalability, clustering, and memory analysis may be essential as the Android malware analysis approach with the annual increase in the sophistication and explosion of different Android families. The obfuscation of the botnet made the Android security threat more formidable. The mode of operation of NotCompatible.C malware (as shown in figure 2-5), shows its propagation strategies.

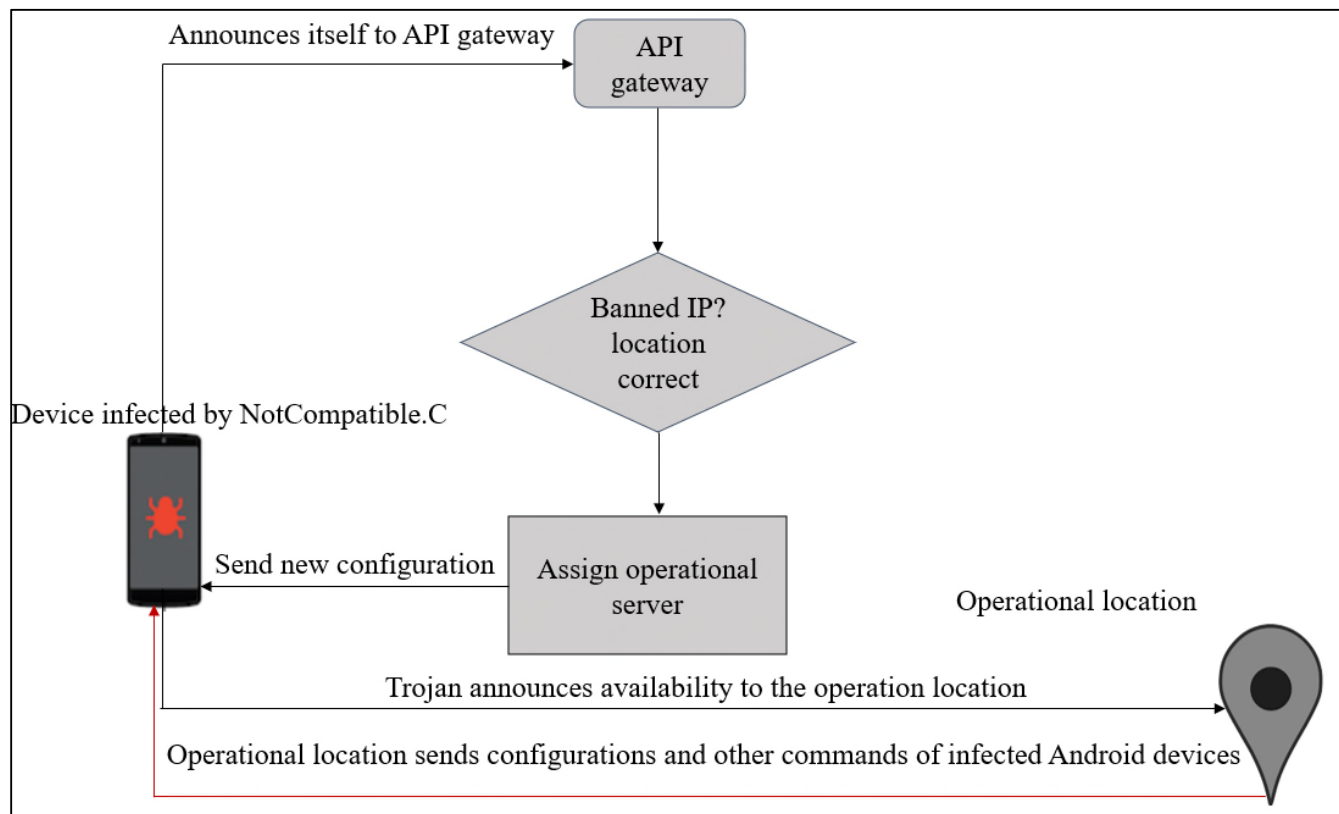


Figure 2-5 Illustrating mode of operation of NOTCompatible.C Android malware

2.3.2.1 Dark web advancements and new malware variants

In 2015, different Android malware emerged such as Spy, Simack, Jssmsers, Benews, Braint-est, Feabme, and Xbot [60]. However, an unknown Android Trojan is comparatively new and different from the former ones, which focuses on harvesting banking details via SMS interception. The Trojan specializes in sniffing banking-related SMS, emails, and forwarding the sniffed data to a hard-coded phone number and email service based in China. While analyzing the intercepted SMS, keywords such as balance, pay, bank, validation, and check are sniffed as part of the data collection and information gathering process. Several Android mobiles infected by the Trojan experienced brute-force which enabled the Trojan to take advantage of the infected devices to exploit the security vulnerabilities and collect sensitive information. Some researchers revealed that user privacy as mobile numbers stored in the database and address book faced security threats when stolen by hackers [61]. Cybercrimes such as identity camouflage become easy to perpetrate when the user's geographical location and phone number are leaked. Also, many mobile banking apps require the owner's mobile number during registration associated with their location.

As the Dark web technologies advance, diverse means are provided where criminal actors (including malware attackers) engage in secure communications and transactions of malicious toolkits and other products and services. Android malware continued evolving with different attack strategies and evasion techniques. Android malicious Buyers and vendors began to exploit the security and business mechanisms on Tor using a multi-signature transaction and encryption approach to spread malware. Secure communications and transactions on malicious applications as software packages became a business of interest on forums and vendor online shops [62-64]. Different Android malware families include Rumms, Krep, Triada, Des-carga, Mazar_Bot, Rootnik, AndroidOS.Fusob, and DroidJack emerged in 2016 [65-67]. According to the mobile security report by Mead et al. [68], AndroidOS.Fusob Trojan ransomware was the most popular and persistent android malicious program in 2016 that infected users in different geographical locations including the UK, US, China, and Germany. Over 230 countries were attacked by Android malware with varying degrees of threat and impact. Some of the stolen information was sold in the dark web market.

One interesting observation is that Android malware began to evolve to infect wider geographical locations, unlike the previous variants and families. Hackers began to decompile legitimate applications and repackage them with embedded malicious code to sell to the marketplaces and increase financial gain. Android malware evolved to a more sophisticated level when AndroidOS.Fusob and DroidJack were used as RATs and can survive a factory reset. These variants were used for state-sponsored attacks and espionage according to Li et al. [69]. Due to their open-source nature, leaked sourced code, and special plugins, the Fully Undetected Remote Access Trojans (RATs) market began to mature with this evolvement of Android malware.

2.3.2.2 Smart malware evasion trends using technological techniques

Android malware took a new dimension when the repackaging of Android applications became very popular in official and third-party markets in 2017. Hackers began to repackage benign applications using reverse-engineering techniques. Consequently, trust in an application has diminished due to repackaging techniques by hackers. Trust in applications influences mobile users' decisions on which market store to download applications from. Mobile vendors must trust that mobile users get legitimate applications during purchases on the marketplace.

According to [70], the Dark Web became a web of cybercriminals where Android and other RATs are sold openly for high prices on Clearnet but with a lower price on Dark Web. Kaspersky reported that in China, AndroidOS.Fusob changes the PIN code and enables Android safety function by resetting the infected device PIN to their Passcode. Android malware continued to evolve through 2016. Different Android malware types significantly grew, especially with the growth in RiskTool. According to Kaspersky, RiskTool files increased from 29% to 43% in 2015 to 2016, respectively. According to the report, Trojan ransom experienced the most significant growth in 2016 with an increase of 4%, which was nearly 6.5 times higher than in 2015.

Trust must be a fundamental component of mobile applications. Mobile applications are being used daily for transactions and other inter-personal collaborations and interactions. However, the trustworthiness of these services and applications, especially in the market store, is still a concern that needs thorough security implementations and checks. The study by Khanmohammadi et al [71] highlighted that the per cent of repackaging android applications grew to 54.38% in 2017 with permissions added in the repackaged applications. The number of malware attacks on Android mobile devices continued to rise from the beginning until the end of 2017.

According to Kaspersky Lab security bulleting [72], Android malware increased about 1.2 times more in 2017 than in the previous year. According to the report, rooting malware is the biggest security threat Android users experienced in 2017, and the percentage of its effects keeps increasing. Super-user rights were gained by the rooting malware (Ztorg) and system vulnerabilities were exploited. The rooting malware was distributed through the play store, which infected and modified over 100 Android applications. This malware family did not affect many devices due to the limited number of Android devices running older versions.

While 230 countries were affected in 2016, the rooting Trojan infected only 161 countries in 2017, according to the Kaspersky report. To keep pace with the security threats presented by potential harmful applications (PHAs) [73], Google introduced multiple layer security architecture during its review of Android security in 2017. The multiple layers of the security and protection architecture advanced to form Google play protection. Though Google play protection has been a fundamental security feature in Android, other features were added in 2017 for better identification of potentially harmful applications on Android devices. Features such as regular updates of Google play were introduced so that the update no longer depends on Over Air updates (OTAs) for security improvement. Our research, however, observed that Google play protect was not sufficient for the total protection of PHAs because its functionality was only on Android devices running 4.3+ versions.

Devices running lesser (older) versions were still subject to PHAs and application exploits. In summary, while the number of Trojan-SMS decreased significantly in 2017, Trojan-Ransomware increased by 5.22% in 2017 compared to 2016. In 2018, Android malware took a different dimension of attack, infection, and propagation strategies, though a built-in mechanism for defence against PHAs was enhanced in the previous year. Potential Harmful Applications infected mobile platforms that installed applications from outside the play store eight times more than those from Google play. While Google Play Protect played a significant role to reduce the rate of mobile infection in 2017, other protection mechanisms such as application sandbox [74], hardened APIs [75], discrete tamper-resistant [76], and BiometricPrompt [77] were developed in 2018 to bring about a reduction in Android malware attacks. The introduction of these security mechanisms in conjunction with platform improvements of Treble [78-79], original equipment manufacturer (OEM) agreements and Android Enterprise Recommended has significantly advanced and improved the Android ecosystem security.

Despite all the security advancements, 0.11% of Android mobile devices were compromised by user-wanted (UW) PHAs called Chamois [80] in 2018. Other variants such as Snowfox, Cosiloon, BreadSMS, View SDK, Triada, CardinalFall Eager-Fonts, and Idle Coconut [81] later emerged with similar characteristics. Chamois was, however, judged to be more sophisticated and injected several sideloaded applications because it infected over 20 million android users. As demonstrated in Figure 2-6, the backdoor generates invalid traffic, performs artificial application promotion, telephone fraud, and dynamic execution of additional plugins.

The User-wanted PHAs disabled SELinux and root Android devices by disabling the SELinux security feature. SELinux disablement led to the increasing of pre-installed PHAs and increased threats of backdoor SDKs. Also, the excessive privileges permit potential harmful applications to defend themselves against any attempt for removal by users. Backdoored SDKs code injected into genuine Android functionalities significantly enhanced the propagation of pre-installed PHAs that compromised Android device integrity, click fraud and attribution fraud. Compared to applications downloaded outside of the play store, the rate of PHAs was high compared to those from Google play (as shown in figures 2-7).

Android malware continues to increase in both scope and complexity, despite the security implantation in the Android operation system. The more the security experts deploy techniques to detect and prevent attacks, the more adversaries deploy alternative and innovative strategies to adapt to the current security situation. This was evident when a sophisticated Android malware family known as TimpDoor emerged in 2019 with unique attributes and rapid propagation and threats. TimpDoor mal-ware bypasses the play store by infecting Android victims through SMS. According to the McAfee threat report [82], the TimpDoor malware family proves extremely effective by infecting more users than the older known Android malware families. The spyware exfiltrates victims' contact, photos, and SMS. The evolving functionality observed of Timp-Door is that the malware uses a SOCKS proxy that redirects traffic. The most probable attacks added by this evolution are click fraud and DoS attacks.

The most worrying characteristic of this threat is the use of filched data such as photos and phone contacts to create false accounts on social media and other online services to steal more identities of users and attack Android-based IoT devices. As shown by Kumar et al [83], the Android platform's integration with IoT-based applications facilitates malware propagation. IoT revolution has covered different spheres of life including remote monitoring, the healthcare sector, and many

others create attack platforms for extensive exploitation of Android-IoT devices. This is a result of the malicious data storage in the IoT blockchain which most of the time is undetected and hard to access for analysis. The study by Taylor et al [84] affirms that malware data deposited in blockchain history are easily transferred via the network and the propagation attack surface enlarges, thus infecting many devices. Though TimpDoor was more sophisticated, LeifAccess Android malware impacts were significantly felt in the USA more than in any other country.

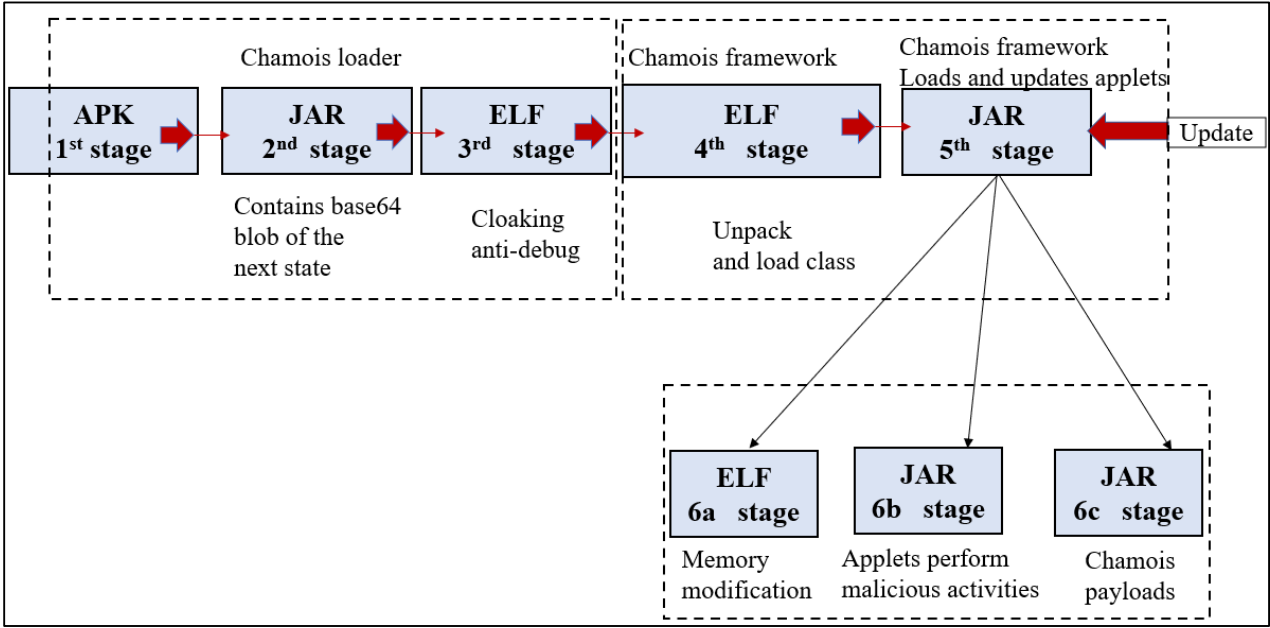


Figure 2-6 Chamois backdoor operational propagation stages with each encrypted and obfuscated

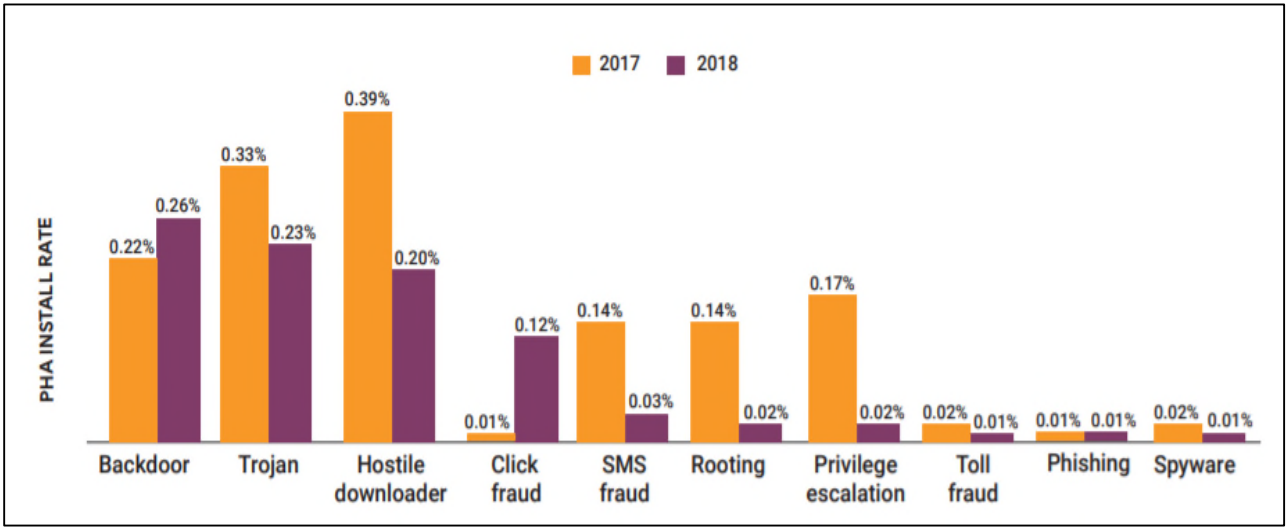


Figure 2-7 Comparison of PHA categories distribution outside of Google Play in 2017 and 2018 [75]

2020 witnessed the tremendous effects of COVID-19 that resulted in lockdown worldwide. While the threat actor strategies and procedures are the same, organizations' exposure to risk levels has greatly increased due to slow response and recovery approaches during the pandemic. As employees continued working from home due to the COVID-19 pandemic, different variants of Android malware emerged. Most employees prefer to use their mobile devices for personal and business transactions under a poorly protected home network. According to the Deloitte security report [85], the contact tracing application launched by the NHS has become a threat actor for social engineering leverage that tricks users into installing fake and malicious COVID-19 contact tracing apps. According to the report, the fake COVID-19 application is being used to deliver different threats.

A new Android mobile malware Unicorn emerged in Italy that targets doctors, pharmacies, businesses, and universities. The ransomware uses social engineering to encrypt users' devices and requests a ransom in euros. Other malicious threats such as email spam, fake landing pages, phishing, and so on emerged when four other variants of the same malware surfaced later. One of the major evasion strategies adopted by recent malware is the Trojan defence approach, as identified in the security report of ESET [86]. Table 2-2 summarises Android malware evolution from 2010 to 2020. Based on the papers examined as part of this study, we classified mobile malware evolution into five (5) categories based on sophistication, characteristics, and intents:

Category 1: This group of mobile malware is characterized by annoying pop-ups and ads usually displayed on the mobile screen. The propagation mechanism is usually via SMS, MMS, and Bluetooth. The intent is for fun, behavioural software testing, knowledge testing, or both.

Category 2: This group of mobile malware is characterized by information theft and privacy violation. The propagation mechanism is usually via social engineering, drive-by download, and key-permutation. The intent is for knowledge testing and financial gain.

Category 3: This group of mobile malware is characterized by device rooting, jailbreaking, and rickrolling. The intent is for financial gain and physical damage to the mobile device. The propagation mechanism is usually via the third-party application and marketplace.

Category 4: This group of mobile malware is characterized by remotely accessing mobile devices. The intent is for financial gain, industrial espionage, vengeance, organization of state-sponsored attack, buffer overflow, remote monitoring, installation of other Trojans, spying, and stealing corporate secrets. The propagation mechanism is usually the internet.

Category 5: Our research referred to this group of mobile malware as modular malware. This group of mobile malware can gain administrative rights of a device and can perform a DDoS attack. The propagation mechanism is usually the internet.

Table 2-2 Android malware evolution.

Year	Prominent malware	Malware type	Kind of havoc	Mode of propagation	Description
2010	AndroidOS. DroidSMS.A	Trojan	Premium SMS	Botnet	A Trojan sends an SMS and charges the victim without his consent
2010	Tap Snake	Spyware	Location monitoring	Native code execution	Sends the device location with the recorded phone conversation to a remote malicious server
2011	DroidDream	Trojan	Device rooting	Code encryption	Roots devices and steals sensitive information from victims
2012	Boxer	Trojan	Premium SMS	Via SMS	The Trojan automatically installs once the victim receives SMS
2012	Opfake	Trojan	Root-level exploit,	Drive-by-download	Performs update attacks
2012	Fakeinst	Trojan	Update attack	Key permutation	Performs update attack. Compromises devices and makes them more vulnerable to more attacks
2013	FakeDefender	Ransomware	Social engineering	Drive-by download	Prompts the user to buy a security app
2013	Obad	Backdoor	Zero-day attack	code obfuscation	Obad exploits zero-day weaknesses to gain more and high privileges on the infected devices

2014	NotCompatible.CA	Trojan	Device side loading	Drive-by download	Self-protecting and persistence through encryption and redundant activities. Aware of the emulated and sandbox and evades detection. Sideloads apps to hinder security assurance
2015	Acecard	Trojan	Apps Overlays	repackaging	Steals banking data of the victim
2015	888.apk	Trojan	Brute force	Repackaging	The Trojan specializes in sniffing banking-related SMS, emails, and forwarding the sniffed data to a hardcoded phone number and email service based in China.
2016	HummingBad	Trojan	Rootkit	Drive-by download	Establishes persistent rootkit
2016	Xbot	Ransomware	Locks screen and files	Code encryption	Steals banking credentials send premium SMS
2016	AndroidOS.Fusob	Trojan-ransomware	RATs	Drive-by download	Remotely access infected devices. demands ransom
2017	ToastAmigo	Backdoor	Apps Overlays	repackaging	Deploys toast overlay attack to install more malicious apps
2017	Ztorg	Trojan	Device rooting	Play store	Super-user rights were gained by the rooting malware (Ztorg) and system vulnerabilities were exploited. The rooting malware was distributed through the play store.
2018	Chamois	Backdoor	RATs	Dynamic code loading	The backdoor steals OAuth tokens. Significantly enhances the propagation of pre-installed PHAs that compromised Android device integrity, click fraud, and attribution fraud.
2019	TimpDoor	Spyware	Click fraud	Java reflection	TimpDoor malware bypass play store by infecting Android victims through SMS
2019	Cerberus	Trojan	Apps Overlays	Drive-by download	Intercepts calls
2019	XHelper	Trojan	Premium SMS	Drive-by download	Displays popup adds, redirects users, send premium SMS
2020	Ghimob, Unicorn	Ransomware	Encrypts files	Drive-by download	The ransomware malware adopts a Trojan defence approach to evade detection

2.4 Classification of mobile malware

Smartphone development and its popularity have brought about a surge in the number of mobile variants. Despite anti-malware advancement solutions, sophisticated malware variants use techniques such as code encryption, code obfuscation, among others to evade detection. The advent of COVID-19 has accelerated mobile and PC variants, because most workers work remotely using unsecured and weak security networks with no physical monitoring and the help of IT support teams. As businesses face economic insecurity due to the advent of the pandemic, the number of cyber threats such as mobile malware is also increasing correspondingly, as highlighted by Brown [87]. It will be unjust to expect employees who work at home using the home network with weak security to be able to protect themselves and their organizations from being attacked by malware without understanding malware variants. Our review examined some of the malware variants and their modus operandi.

2.4.1 Ransomware

Although ransomware is commonly experienced on computers, mobile devices are not immune from being infected with this type of malware. Ransomware attack locks the mobile screen and encrypts files with a displayed ransom message, usually demanding a Bitcoin payment. Android ransomware is currently increasing, and it is of paramount significance to have a defensive approach that will guarantee the data security of mobile users. Most of the defensive approaches are signature-based and, thus, ineffective for the current state-of-the-art malware variants. Mobile ransomware is a malicious variant whose design and operation block entrance or access to a mobile device until the amount of money is paid. Recent research has shown that 74% of companies in the 21st century are under terrific ransomware siege, especially those whose operations are connected to mobile devices [88]. Mobile ransomware harvests data such as photos, videos, usernames, and passwords related to a financial transaction.

2.4.2 Adware

Initially, adware was known for the annoying and frustrating pop-up characteristics without any specific malicious intent. However, it has now moved from the realm of just pop-up displays to data collection. Some adware goes as far as rooting (for android) and jailbreaking (for Apple) mobile

devices, while others have the capability of draining the device battery when not observed and prevented. Mobile adware such as Cydia [89] steals user data and modifies mobile IMEI, IMSI, and steals android mobile Transaction numbers (mTAN). Mobile adware has evolved from creating banners and ads to easy and un-noticed mobile rooting and jailbreaking.

Some create adverts as a link to lure victims into malicious sites for auto-download of malicious applications once visited. For instance, a banner or an ad may appear on your phone screen while online with the statement ‘Amazon is giving out £350 worth of voucher cards, click the link below to claim yours. This trick harvests sensitive data used to carry out sophisticated attacks. These details are then used to launch an attack on the victim’s mobile device. Some adware variants create backdoors on mobile devices when infected to facilitate further and future attacks. Some adware could deploy social engineering tactics to lure the victim into divulging sensitive information that is after that used for the attack.

2.4.3 Trojan

Android Trojans are camouflaged as legitimate Android applications. Trojans harvest sensitive information on a device, spy on the user's activities, delete the user’s files and can download other malicious activities. The study by Imtiaz et al [90] reports that some Android Trojans masquerade as legitimate mobile banking applications, mostly targeting android mobile phones. The research reported that an Android mobile unnamed Trojan “888.apk” intercepts and sniffs mobile banking transaction packets during SMS alerts and transaction commands such as Check, Validation, and pay as an attack banner grabbing process. Apart from the 888.apk mobile Trojan, Kaspersky reported the emergence of a new Android banking Trojan Svpeng known as Trojan-SMS.AndroidOS.Svpeng. Svpeng steals mobile banking credentials such as passwords and usernames upon launching a mobile banking application. The Trojan targeted only Russians using android phones and later people in other countries. Mobile Trojan horses are a dangerous set of mobile malware which in most occasions appear fictitious as beneficial applications but with hidden malicious content and action when executed. Some mobile Trojans can obliterate a whole mobile or computer drive when infected, while some serve as a backdoor for device remote control.

Malware writers apply the principle of social engineering as a means of gaining financial benefits from victims using SMS premium service. Through this strategy, victims subscribe to certain services or applications without their consent. Scammers usually use users' phone numbers or emails to execute this malicious scheme. Most malicious premium messages to victims seem very real and appealing such as: 'Know who blocks you on WhatsApp' 'Click here to see the message sent to you by a friend', 'Install battery speedometer to see how long your battery lasts', and 'Amazon is giving a free £25 to prime members'; click here to claim yours'. Hackers make these tricks look appealing and real to entice the victim. Once the phone number is supplied, an automatic subscription to your monthly mobile bill is activated for any SMS received or sent.

For advanced SMS premium service attacks [91], the victim does not have to be tricked into giving his phone number. Hackers' banners grab the social networks account details of the victim, specifically his phone number. Therefore, it is advisable to critically look at the content of any email sent to you before opening it, especially if it looks suspicious. More importantly, it is worth not trusting easy financial claim links sent to you if not from your financial institutions such as banks and cooperatives. It is pertinent to note that the SMS premium service charge attack cannot be achieved if the malware does not have permission to access the device service "send Text Message", which is the mobile privacy permission SMS service. Hence, understanding permission requests are very significant in fighting mobile SMS Trojans.

2.4.4 Rootkit

A rootkit is a mobile malware type that controls and exploits a mobile device through remote access. A rootkit comprises a loader, rootkit, and dropper. The rootkit gains administrative access and installs other applications at the backdoor that are malicious without the consent of the victim. Rootkit alters the configurations of a device once installed. Due to their silence and underground operations, it is very difficult to identify and remove the rootkit. Evasion techniques such as obfuscation are adopted by roots to make it stay undetected in a device for a long time. HummingBad [92] is an example of a rootkit that steals credentials by installing other malicious applications in the background to create fake ads.

2.4.5 Botnets and viruses

Botnets are compromised devices whereby an attacker remotely accesses and controls the infected devices. The botmaster controls the infected mobile phones without the knowledge of the victims. Botnets have become a serious threat to information and mobile security, as attacks such as DDoS are launched using mobile botnets. Double-Door [93] is an example of a popular mobile malware botnet. Mobile viruses are small computer codes built to attack devices operating in cellular environments, such as mobile phones and PDAs. They are computer programs whose designs are borne out of curiosity to gain general attention and target and exploit vulnerable mobile phones and other applications with a high degree of versatility. A mobile virus is propagated the moment an infected file or application is executed, causing a speedy escalation to connected devices and other application segments. The ruinous consequences of a virus on mobile devices and applications range from data loss, application loss, and device destruction on most occasions, if not discovered early. Some viruses evade detection and may be difficult to be identified by traditional detection mechanisms.

When the running OS of memory in a device (mobile or computer) is infected with virus code, all the programs including the executable code running in the core memory of such a device can also be infected including its internal storage Figure 2-8. The virus program first requests permission from the device OS to attach its code to the device object module, especially at the system start-up. After the septic virus code is invoked in the device memory, the virus then switches over and takes over the entire system. The infected code then spreads to the device operating system. In the case of a computer, a floppy disk serves as the object code, which helps in virus invocation and execution at system start-up.

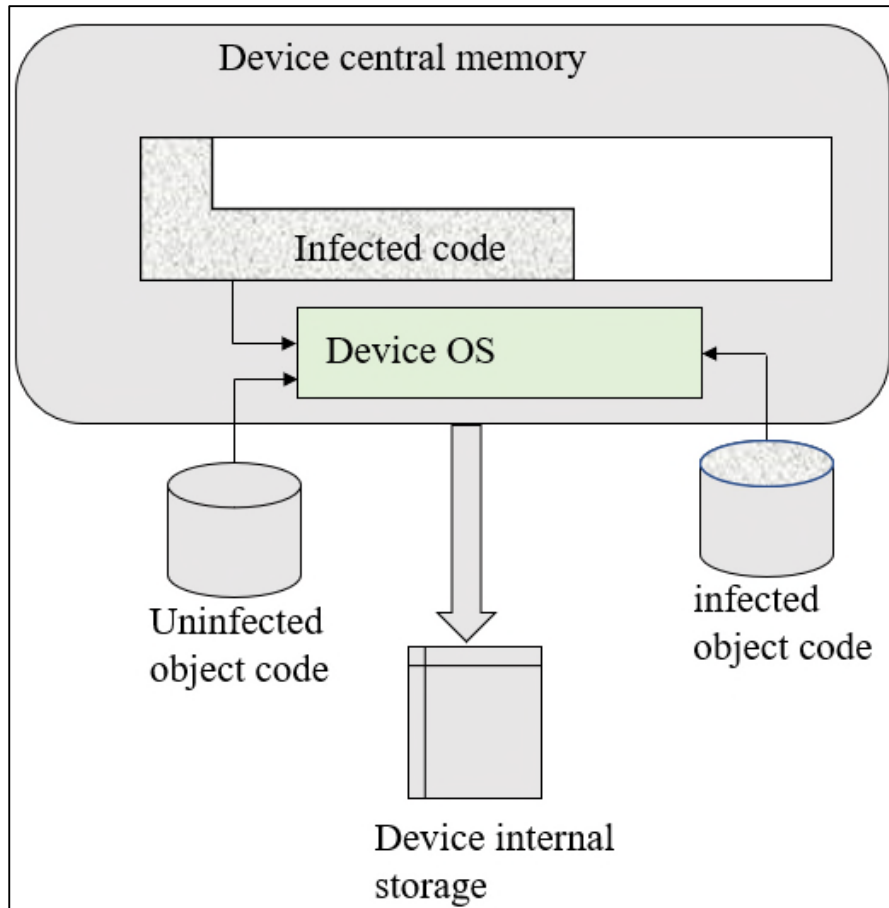


Figure 2-8 Mobile virus infection strategy

2.4.6 Worms

Android worms are malicious codes that black hackers develop for malicious intents and operations for mobile devices. Worms have the capability to self-replicate from one susceptible mobile device to another with little or no external trigger such as human behaviour. Mobile worms assume different file formats, such as pdf and file extensions, to invade a vulnerable device. The peculiar infection character of worms is that the payload executes once on the initial device and later escalates to other targeted devices via attack vectors such as TCP, email, IP, SMS, etc. Based on the papers examined as part of this study, we classified mobile worms into five (5) main categories with their major characteristics as follows:

Binary file worms: These are forms of worms that infect device executable files. They are usually programmed in machine language for easy payload distribution.

Multi-partite worms: These are worms with the capability of affecting mobile and computer boot sectors and executable files. This form of worm is rare.

Script file worms: These are worms technically written in human-readable form, which requires translation, by an interpreter to perform machine-executable.

Binary stream worms: The infection vector deployed by this form of worms is pure via a network connection. It relies on the device to be linked to the network before implementing its infection.

Macro worms: These are worms that infect applications and data files such as documents. Macro worms seem to be the most common. To reduce worm infections on Android devices and mobile platforms in general, it is pertinent to regularly upgrade the mobile OS to the latest versions to avoid mobile susceptibility to signature-based and zero-day attacks. A summary of the malware classes and their distinct but related behaviour is presented in Table 2-3. Though the use of the latest antivirus engines is very significant, it is recommendable to ensure that the latest mobile device patches are constantly installed and updated. By this, the worm infection rate and worm propagation can be considerably reduced. This strategy is essential because our research believes that the worm's infection rate on mobile devices is directly proportional to the propagation speed. An increase in worm removal on affected devices can also subdue its propagation, thus reducing the number of mobile devices that can be contaminated. Other types of Android malware include Spyware and key loggers. Android OS fragmentation and open source have created security loopholes resulting in the great increase in attacks we are seeing now.

Apart from the entire android platform, the majority of the android applications have untrusted digital signatures within their security parameters. While Apple has a proactive approach to malware and general threats, Google is more concerned about taking reactive defence approaches when a threat occurs. In addition, while iOS sticks to Xcode [94] with default IDE choice with minimal platforms which are maximally managed and embedded with better security primitives, Android has several platforms with little security entrenchment. These factors made Android devices the devices most infected by malware. The report produced by Li et al [95] affirmed that two-thirds of mobile malware targeted Android platform devices. This shows the degree of android vulnerability to malware. It, therefore, calls for appropriate security strategies to address this problem timely.

Table 2-3 Mobile malware variants with their distinctive characteristics

Mobile Malware variants	Characteristics
Mobile Ransomware	Locks out users' documents, encrypts files, requests for ransoms usually inform of bitcoin
Mobile adware	Creates annoying pop-ups, creates backdoors, auto-download, use social engineering tactics
Mobile SMS Trojans	Does not replicate itself, creates backdoors, may engage in secondary infection, does not infect other files, utilizes social engineering principle for payload execution, uses SMS, Links, and MMS for distribution
Mobile worms	Self-replicating, may have no payload, could be benign or malignant
Mobile virus	Written in small sizeable codes, highly versatile, exceedingly reoccurring during propagation to other devices, effective in data loss and device damage, a wide variety of functions
Botnets	An attacker remotely access and control the infected devices
Rootkit	Gains administrative access and at the backdoor install other applications

2.5 Mobile malware infection vectors

Infection strategies are attack vectors through which designed mobile malware access the target device to execute their payload. Malware files remain non-executable and uninfected until a path is created via which access to the device could be achieved. While malware was written earlier, its propagation rate was slow compared to recent malware. The speed of malware propagation is significantly enhanced by internet advancement. Internet connectivity is one of the major routes via which malware payloads are easily transmitted to other subsidiaries such as Bluetooth, SMS, cellular networks, Wi-Fi, physical access, and USB-PC connections.

2.5.1 Bluetooth/SMS distribution

Mobile malware such as Commonwarrior was discovered in 2005 and spread through Wi-Fi and SMS as an attack vector. As cited in Figure 2-9, mobile malware spreading through this medium has a large spectrum of infecting an entire continent, since there is nothing to restrict the spread except the exhausted balance of the user's device. Pebler mobile is known to infect mobile phones by exploiting vulnerabilities found in Wi-Fi and network connectivity to devices. The only limitation of this attack vector is that mobile devices with no internet features suffer little. As technology advances, mobile devices with embedded Bluetooth technology can communicate and share files with nearby devices, which are a few meters away. Some Trojans exploit Bluetooth to spread to other devices. The danger of this attack vector is that although it can only infect nearby devices, it can cause a devastating septicity where many Bluetooth devices are enabled [96].

Bluetooth attack vector has a high threshold of spreading malware since the majority of mobile users share mobile files such as pictures, videos, and audio files without considering the consequences. Advancement in Bluetooth technology to enhance the location down to centimetre-level precision has created and expanded mobile malware propagation conduit. For instance, the inclusion of a point of interest application (POI) [97] for information solutions enhances the speedy replication of worms using Bluetooth. SMS is another mobile malware distribution strategy where malware payload is being disseminated to mobile or devices through SMS or MMS. The propagation could either be through sending SMS or MMS from one mobile device to another or from the cloud to the mobile device. Malware uses SMS as a distribution attack vector to mobile devices where the target device is infected to send illegal and unauthorised texts, and the victim is directly charged for the call or text message.

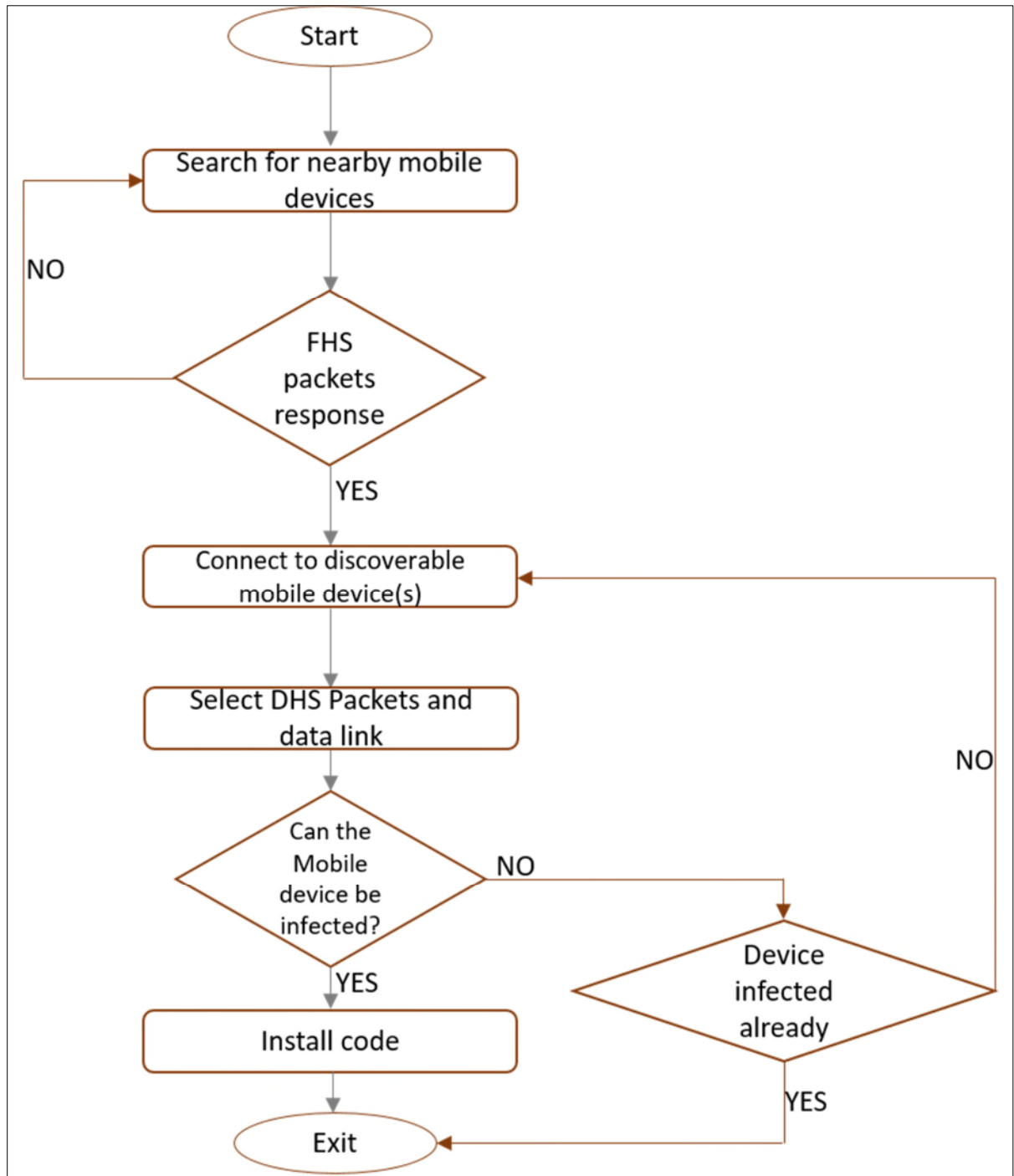


Figure 2-9 Malware propagation using Bluetooth technique

2.5.2 USB/network to device distribution

Apart from the Bluetooth infection strategy, USB is one of the distribution strategies. Different developers upload and update their malicious codes on the Apps store, camouflaging them as genuine software. When the user installs a malicious application, malware unknowingly infects the user's device. When mobile devices connect to PCs through USB, this offers the opportunity for a malware-infected PC to transmit to the victim's device. The attack vector serves as a thumb drive for PC-Mobile malware infection. This is where malware is distributed from Universal Serial Bus (USB) to any device, which could be PCs or mobile devices. As was announced by IBM, many USB sticks with the following product model "2071, 02A and 10A are infected with malware which has access to the device via USB cables or port when a connection is established between the USB and the device. When a mobile device is connected via a USB port to an infected computer, malware distribution to the phone becomes inevitable. This distribution technique is used by malware in the form of ads or applications connected to the internet and masquerading as real applications.

2.5.3 Market/application to device distribution

Application to Device known as A2D is a distribution mechanism where the mobile malware depends solely on the application vulnerability to distribute itself. An example of android malware, which exhibits spreading and infection vector via A2D approach, is Andr/Opfake [98]. In the Market to Device strategy in distributing malware, a malicious application is uploaded into the application store market. The infection by users is dependent on the application reception in the market and users' installation on their mobile phones. On a general note, some of these infections happen when users make conscious resolutions to carry out an act. Behaviour like scanning barcodes, QR codes, or logging into an unsecured compromised Wi-Fi network is an example that led to users' mobile devices being exploited by malware.

In summary, we assume a malware type could infect a susceptible mobile device after effective communication with a contagious device if an immunity to that same type of malware is not developed. Based on the meta-analysis of the infection strategies, we assume that when a mobile device becomes infected, it can develop immunity to that same type of malware and can recover if adequate security strategies are applied immediately. However, such a device will remain

vulnerable to other variants of the same malware after recovery. Also, an infected device can spread the malware if not kept under control.

2.6 Classification of Smart malware based on behavioural

Different criteria are used in classifying mobile malware such as classification based on device-platform [99-100], family [101], file system [102], resource consumption [103], and many other criteria. Our research uses behavioural attributes as a major classification criterion of mobile malware. Under behavioural attributes, our research focused mainly on privilege escalation and attack goals as discussed in the subsection. Malware privilege escalation has attack goals. For instance, people downloading pornographic files do so to gratify their carnal urges, just like those who indulge in online provocation and harassment do so to have power over the other party. In addition, just like any other cybercrime, the goals behind every cybercrime are financial incentives, power greed, adventure, and vengeance. Some malicious activities on mobile devices are to steal sensitive personal credentials, while others are for SMS premium where charges are made without embarking on any service or SMS by the user unknowingly.

2.6.1 Classification based on privilege escalation

Privilege means what a device user is allowed or permitted to do on such a device. Examples of such include file editing or modifications. In the context of our research, it is viewed as the art of malware exploiting a vulnerability, bug, configuration, or design flaw of a mobile device or application to have enough unauthorized privileges to access mobile resources. When this occurs, malware reads and writes to files, may insert or attach a permanent backdoor (as shown in figure 2-10). Malware exploits some strategies to gain the required privilege. Some of these strategies could be user manipulation or technical exploitation. The technical exploitation path usually followed by malware to gain escalated privilege to mobile devices is through platform misconfiguration or technical susceptibility. Malware technical exploitations modify the security restrictions of a device. Once this is accomplished, exploiting other vulnerabilities on the affected system becomes very easy.

This technical exploitation malware imbibes to achieve privilege escalation including buffer overruns, session attacks, SQL injection, cross-site attacks, file disclosure system vulnerability, and networking configuration flaws. The attack behaviour and goals include sabotage, spam, and service misuse. This analysis clearly shows that mobile malware displays different behavioural sets. It is worthy to note that the mobile malware attack goal has different categories of incentives, which are interwoven. It means that an attack goal can have more than one motivation. For instance, SPAM as an attack goal has both personal information theft and financial profit as motivation types.

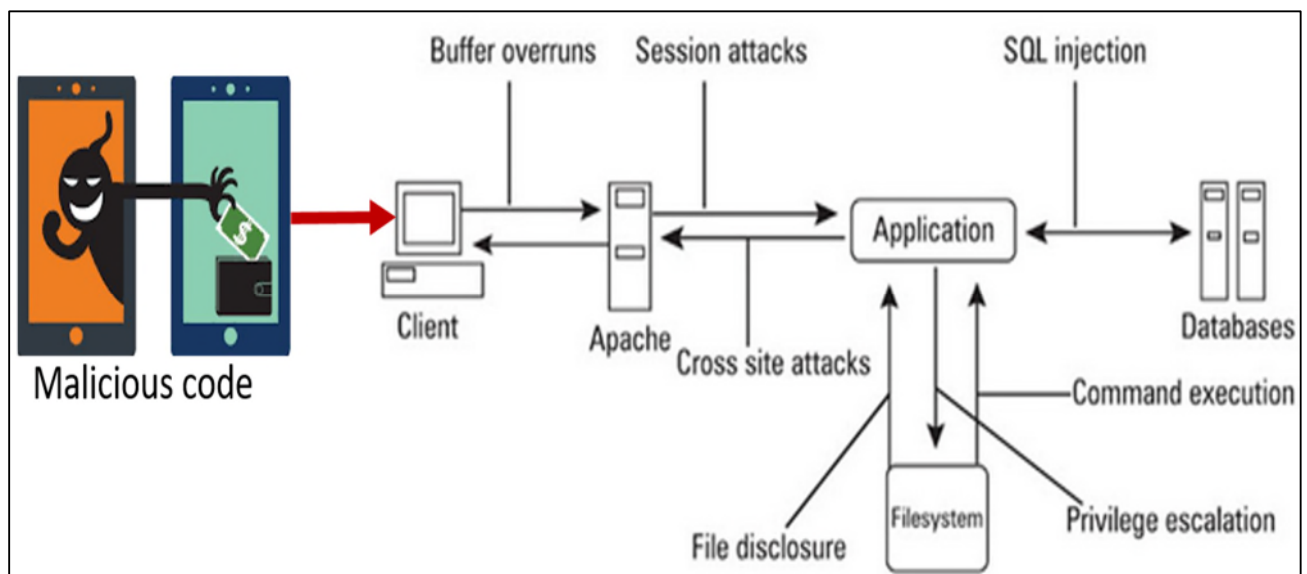


Figure 2-10 Mobile malware Privilege escalation illustration

2.6.2 Classification based on attack goals

Though malware keeps evolving to take different attack trajectories, their attack goals and motivations remain the same (see figure 2-11). While some malware writers could focus on stealing sensitive information as a primary step upon which subsequent attacks could be relied upon, others focus on the actual attack goal in their code implementation. Our research classified mobile malware attack goals based on their motivation type and behaviour related to the motivations. Table 2-4 shows the motivation types and major attack goals classified in our research. Cybercriminals perform attacks to achieve one form of intention or the other. This means that for every attack, there is an equal motivation behind it. Just as malware writers develop malicious applications for different motives such as financial profit, identity theft, industrial espionage, and a host of others,

people that use these applications also have motives behind their usage. Based on their characteristics, mobile malware deploys different infection strategies based on the attack goals.

We observed that the motivations behind every mobile attack are different: financial, power, greed, adventure, and vengeance. While some malicious activities on mobile devices target sensitive personal credentials, others are for SMS premium where charges are made without embarking on any service or SMS by the user unknowingly. The attack behaviour and goals include sabotage, spam, and service misuse. These are pure types of motivations for mobile malware attacks. The attack schemes that formed the behaviour related to those motivations could include eavesdropping, profiling, and loggers. Kocher et al [104] show how to pattern malicious behaviour using security inspection strategies. This confirms that mobile malware exhibits different attack goals during execution. It is worthy to note that mobile malware attack goals have different categories of motivations, which are interwoven. This means that an attack goal can have more than one motivation. For instance, SPAM as an attack goal has both information theft and financial profit as motivation types.

The unusual behaviour of an application should be reported promptly. During application download and installations, mobile application plugins and codecs are pointers to be observed that the app is from a legitimate store. Mobile users could deploy software solutions from Enterprise Mobility Management to enhance the security on enterprise devices. These security strategies could help reduce the impact of the rising mobile malware threats.

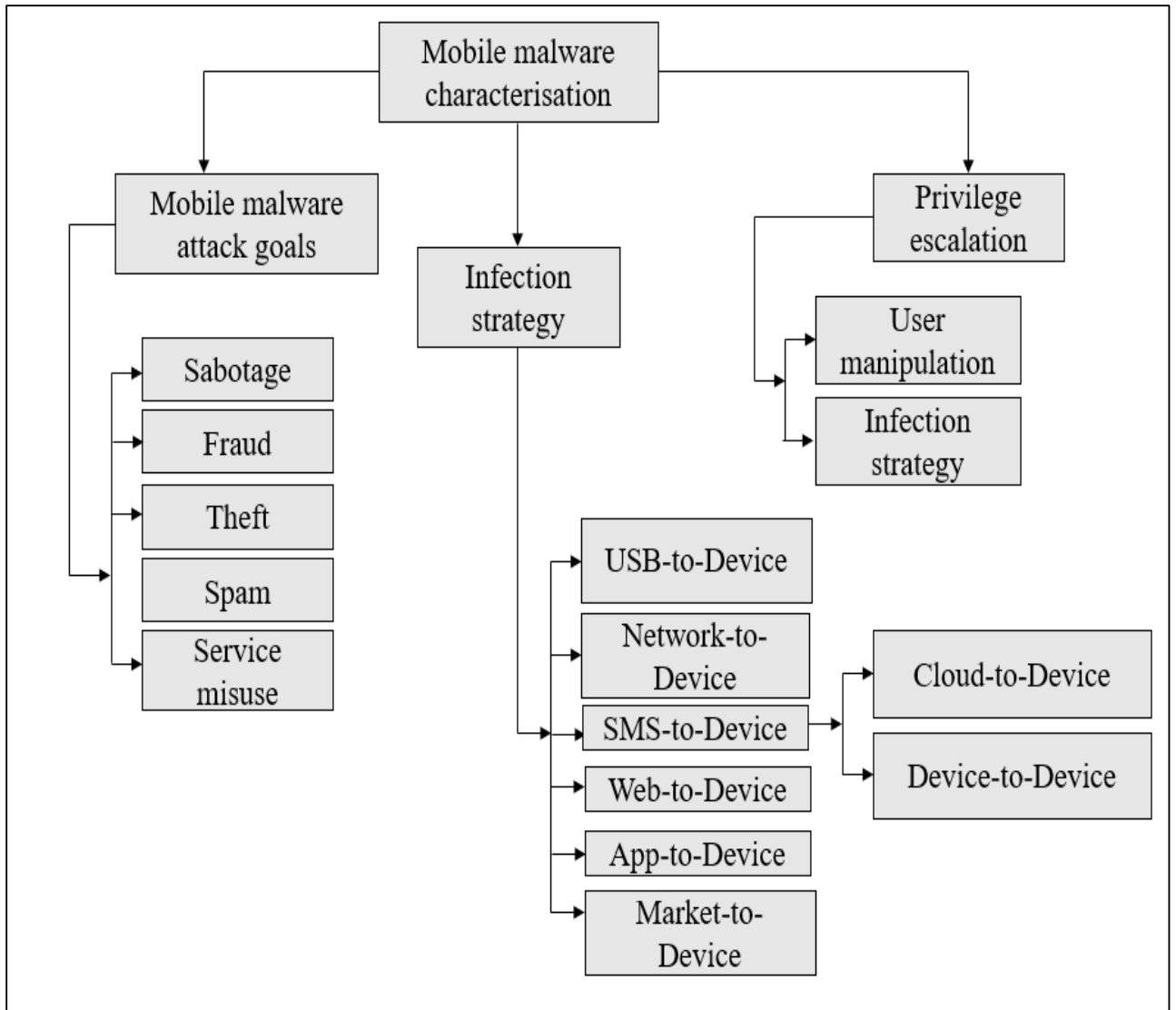


Figure 2-11 Mobile malware characterization

Table 2-4 Mobile malware infection objectives

Major attack goals					
Motivations	Sabotage	Fraud	Data Theft	Spam	Misuse of Service
Types of Motivations	*Financial profit *Industrial Espionage	*Financial profit *ID theft *PI Theft *Industrial Espionage	*Financial profit *Personal Information Theft *Industrial Espionage	*Financial profit *Personal Information Theft	Financial profit
Behaviour Related to the Motivations	*Eavesdropping -SMS -Multimedia *Photo *Audio *Video	*Loggers Touch Key	*Loggers • Touch • Key	* Profiling • Apps • Location	Combination of any of the previous

2.7 Conclusion

Malware attacks on mobile devices are increasing with the increase in the number of mobile applications published on the App and Play store daily. Furthermore, mobile malware infection increases as mobile technology keep advancing. Most of the emerging mobile functionalities constitute the attack vectors malware deploys to infect mobile devices. For instance, the first-generation mobile malware could not spread quickly due to low knowledge of wireless hacking, short Bluetooth range coverage, a small Wi-Fi network population, and the lack of cross-platform propagation. Other factors that enhance the spread of mobile malware include advancement in threat dimensions, improved security, and business communication on Tor using multi-signature transaction and encryption approach, mobile platforms' integration with the IoT-based applications, the emergence of evasion techniques, and the advancements in mobile malware Toolkits on the dark web. To

achieve the aim of the paper, we conducted an up-to-date study of mobile malware evolution and infection strategies. This enabled us to classify mobile malware evolution trends into distinct categories based on their sophistication, characteristics, and attack intents. We identified factors that limited the fast spread of the early mobile mal-ware such as lack of mobile OS standardization, lack of mobile OS cross-platform, and lack of Blue-tooth technological advancements in data communication. Finally, we identified factors that enhanced the fast spread of recent mobile malware. These included the advancement in threat dimensions, improved security and business communication on Tor using a multi-signature transaction and encryption approach, mobile platforms' integration with the IoT-based applications, the emergence of evasion techniques, and the advancements in mobile mal-ware Toolkits on the dark web.

Furthermore, this paper has discussed research conducted from 2000-2020 on mobile malware evolution with their infection strategies. In the end, the paper identifies the need to analyse the existing detection techniques to identify their strengths and weaknesses to help develop more robust and accurate tools for an Android malware detection standpoint. We suggest the need for a malware infection model to prevent mobile malware spread among mobile platforms with random additive perturbations of infection rate. This strategy will help to understand the immunity state of a mobile device during and after recovery from malware infection. The effect of the random perturbation on malware stability behaviour might be essential in determining some transient characteristics of malware infection states.

ACKNOWLEDGEMENTS

This work is supported by the Petroleum Technology Development Fund (PTDF) under Grant PTDF/ED/PHD/AMA/1245/17-17.

REFERENCES

- [1] E. Goh and M. Sigala, "Integrating Information & Communication Technologies (ICT) into classroom instruction: teaching tips for hospitality educators from a diffusion of innovation approach," *J. Teach. Travel Tour.*, vol. 20, no. 2, pp. 156–165, 2020, doi: 10.1080/15313220.2020.1740636.
- [2] Z. G. Shatri, "Advantages and disadvantages of using information technology in the learning process of students," *J. Turkish Sci. Educ.*, vol. 17, no. 3, pp. 420–428, 2020, doi: 10.36681/tused.2020.36.
- [3] L. Maglaras, G. Drivas, N. Chouliaras, E. Boiten, C. Lambrinoudakis, and S. Ioannidis, "Cybersecurity in the Era of Digital Transformation: The case of Greece," in *2020 Int. Conf. Internet Things Intell. Appl. ITIA*, Zhenjiang, China, 27-29 Nov. 2020, pp. 0–4, 2020, doi: 10.1109/ITIA50152.2020.9312297.
- [4] H. S. Lallie et al., "Cyber security in the age of COVID-19: A timeline and analysis of cyber-crime and cyber-attacks during the pandemic," *Comput. Secur.*, vol. 105, p. 102248, 2020, doi: 10.1016/j.cose.2021.102248.
- [5] N. A. Khan, S. N. Brohi, and N. Zaman, "Ten Deadly Cyber Security Threats Amid COVID-19 Pandemic," *TechRxiv Powered by IEEE*, pp. 1–6, 2020, doi: 10.36227/techrxiv.12278792.v1.
- [6] L. M. Cristea, "Current security threats in the national and international context," *J. Account. Manag. Inf. Syst.*, vol. 19, no. 2, pp. 351–378, 2020, doi: 10.24818/jamis.2020.02007.
- [7] P. McLaren, G. Russell, and B. Buchanan, "Mining malware command and control traces," in *Proc. Comput. Conf. London, UK*, 18-20 July 2018, doi: 10.1109/SAI.2017.8252185.
- [8] M. S. Saleem, J. Misic, and V. B. Misic, "Examining Permission Patterns in Android Apps using Kernel Density Estimation," in *2020 Int. Conf. Comput. Netw. Commun. ICNC 2020*, pp. 719–724, 2020, doi: 10.1109/ICNC47757.2020.9049820.
- [9] J. Gamba, M. Rashed, A. Razaghpanah, J. Tapiador and N. Vallina-Rodriguez, "An Analysis of Pre-installed Android Software," *2020 IEEE Symposium on Security and Privacy (SP)*, 2020, pp. 1039-1055, doi: 10.1109/SP40000.2020.00013.
- [10] J. McGiff, W. G. Hatcher, J. Nguyen, W. Yu, E. Blasch and C. Lu, "Towards Multimodal Learning for Android Malware Detection," *2019 International Conference on Computing, Networking and Communications (ICNC)*, 2019, pp. 432-436, doi: 10.1109/ICCNC.2019.8685502..
- [11] Oxera, "Android in Europe: Benefits to consumers and business," Accessed 4 Oct. 2018. [Online]. Available: <https://www.oxera.com/wp-content/uploads/2018/10/Android-in-Europe-1.pdf>
- [12] A. Adekotujo, A. Odumabo, A. Adedokun, and O. Aiyeniko, "A Comparative Study of Operating Systems: Case of Windows, UNIX, Linux, Mac, Android and iOS," *Int. J. Comput. Appl.*, vol. 176, no. 39, pp. 16–23, 2020, doi: 10.5120/ijca2020920494.
- [13] A. Gerber and C. Clifton, *Learn Android Studio. Build Android Apps Quickly and Effectively*, NY, USA: Apress, May 2015.
- [14] S. Mierzwa, S. RamaRao, J. A. Yun, and B. G. Jeong, "Proposal for the Development and Addition of a Cybersecurity Assessment Section into Technology Involving Global Public Health," *Int. J. Cybersecurity Intell. Cybercrime*, vol. 3, no. 2, pp. 48–61, 2020, [Online]. Available: <https://vc.bridgew.edu/ijcic/vol3/iss2/4>.

- [15] M. Adeel and L. N. Tokarchuk, "Analysis of mobile P2P malware detection framework through Cabir & Commwarrior families," in Proc. - 2011 IEEE Int. Conf. Privacy, Secur. Risk Trust IEEE Int. Conf. Soc. Comput. PASSAT/SocialCom 2011, pp. 1335–1343, 2011, doi: 10.1109/PASSAT/SocialCom.2011.243.
- [16] E. Johns, "Cyber Security Breaches Survey 2020," Comput. Fraud Secur., vol. 2020, no. 4, p. 4, 2020, doi: 10.1016/s1361-3723(20)30037-3.
- [17] Marsh & McLennan and Zurich Insurance Group, "The Global Risks Report 2020," 2020. Accessed on 11 June 2020. [Online]. Available: http://www3.weforum.org/docs/WEF_Global_Risk_Report_2020.pdf
- [18] S. Shi and J. Yin, "Global research on carbon footprint : A scientometric review," Environ. Impact Assess. Rev., vol. 89, no. 2, p. 106571, 2021, doi: 10.1016/j.eiar.2021.106571.
- [19] A. P. Namanya, A. Cullen, I. U. Awan, J. P. Disso, and R. Kit, "The World of Malware : An Overview," in 2018 IEEE 6th Int. Conf. Future Internet Things Cloud (FiCloud), 2018, pp. 420–427, doi: 10.1109/FiCloud.2018.00067.
- [20] L. J. García Villalba, A. L. Sandoval Orozco, A. López Vivar, E. A. Armas Vega and T. Kim, "Ransomware Automatic Data Acquisition Tool," IEEE Access, vol. 6, pp. 55043–55052, 2018, doi: 10.1109/ACCESS.2018.2868885.
- [21] T. Szuba and D. Sztuba, "Can Adam Smith's Invisible Hand phenomenon be used for the analysis of Fourth Estate's impact and behavior?," in 2020 Int. Jt Conf. Neural Netw. (IJCNN), 2020, pp. 1–8, doi: 10.1109/IJCNN48605.2020.9207576.
- [22] W. Pranoto, "Malicious Software Analysis," (in Indonesian), CyberSecurity dan Forensik Digital, vol. 1, no. 2, pp. 62–66, 2018, doi: 10.14421/csecurity.2018.1.2.1374.
- [23] S. Levy and J. R. Crandall, "The program with a personality: Analysis of Elk cloner, the first personal computer virus," Cornell University press, New York, pp. 1–8, 2020.
- [24] H. J. Highland, "History of computer viruses - the famous `trio'," Comput. Secur., vol. 16, no. 5, pp. 416–429, 1997, doi: 10.1016/S0167-4048(97)82246-8.
- [25] L. Fu, "Design of hidden communication remote monitoring based on C / C MFC," Proc. - 2019 4th Int. Conf. Mech. Control Comput. Eng. ICMCCE 2019, pp. 589–591, 2019, doi: 10.1109/ICMCCE48743.2019.00135.
- [26] Checkpoint security, "5 th generation cyber attacks are here and most businesses are behind a new model for assessing and planning security," Accessed on 9 Nov. 2018, [online]. available:<https://www.checkpoint.com/downloads/productrelated/whitepapers/preventing-the-next-mega-cyber-attack.pdf>.
- [27] M. Bartolo et al., "Urgent Measures for the Containment of the Coronavirus (Covid-19) Epidemic in the Neurorehabilitation/Rehabilitation Departments in the Phase of Maximum Expansion of the Epidemic," Front. Neurol., vol. 11, no. 23, pp. 1–6, 2020, doi: 10.3389/fneur.2020.00423.
- [28] J. Kaur, "Taxonomy of Malware : Virus , Worms and Trojan," Int. J. Res. Anal. Rev., vol. 6, no. 1, pp. 192–196, 2019.
- [29] P. R. Shah, Y. Shah, and S. Madan, "Mobile Viruses," International Journal of Computer Applications, vol. 5, no.2, pp. 1–7, 2011.
- [30] M. Chikapa and A. P. Namanya, "Towards a fast off-line static malware analysis framework," pp. 182–187, 2018, doi: 10.1109/W-FiCloud.2018.00035.
- [31] S. Pillet et al., "Contamination of healthcare workers' mobile phones by epidemic viruses," Clin. Microbiol. Infect., vol. 22, no. 5, pp. 456–458, 2016, doi: 10.1016/j.cmi.2015.12.008.
- [32] R. Mayrhofer, J. Vander Stoep, C. Brubaker, and N. Kravovich, "The android platform security model," ACM Transactions on Privacy and Security (TOPS), 24(3), pp.1–35, 2019.

- [33] M. R. Khan, R. C. Tripathi, and A. Kumar, "A malicious attacks and defense techniques on android-based smartphone platform," *Int. J. Innov. Technol. Explor. Eng.*, vol. 8, no. 8 Special Issue 3, pp. 361–369, 2019.
- [34] N. R. Techniques, H. D. D. Expose, A. Target, and M. Lucrative, "McAfee Labs Threats Report: December 2018," *Comput. Fraud Secur.*, vol. 4, no. 1, p. 4, 2019, doi: 10.1016/s1361-3723(19)30004-1.
- [35] B. Runciman, "Cybersecurity Report 2020," *Itnow*, vol. 62, no. 4, pp. 28–29, 2020, doi: 10.1093/itnow/bwaa103.
- [36] F. Alswaina and K. Elleithy, "Android malware family classification and analysis: Current status and future directions," *Electron.*, vol. 9, no. 6, pp. 1–20, 2020, doi: 10.3390/electronics9060942.
- [37] M. O. Topgul and E. I. Tatli, "The Past and Future of Mobile Malwares," In *The 7th International Conference on Information Security and Cryptology*, Istanbul, Turkey, 4 Oct. 2014.
- [38] D. Chen, P. Wawrzynski, and Z. Lv, "Cyber security in smart cities: A review of deep learning-based applications and case studies," *Sustain. Cities Soc.*, vol. 66, 2021, doi: 10.1016/j.scs.2020.102655.
- [39] A. Baranchuk et al., "Cybersecurity for Cardiac Implantable Electronic Devices: What Should You Know?," *J. Am. Coll. Cardiol.*, vol. 71, no. 11, pp. 1284–1288, 2018, doi: 10.1016/j.jacc.2018.01.023.
- [40] M. Akbanov, V. G. Vassilakis, and M. D. Logothetis, "WannaCry ransomware: Analysis of infection, persistence, recovery prevention and propagation mechanisms," *J. Telecommun. Inf. Technol.*, no. 1, pp. 113–124, 2019, doi: 10.26636/jtit.2019.130218.
- [41] Z. Zhu and T. Dumitras, "FeatureSmith : Automatically Engineering Features for Malware Detection by Mining the Security Literature," in *Proc. 2016 ACM SIGSAC Conf. Comput. Commun. Secur.*, Vienna, Austria, 24–28 Oct. 2015, pp. 767–778, 2016, doi: 10.1145/2976749.2978304.
- [42] S. K. Sasidharan and C. Thomas, "ProDroid – An Android malware detection framework based on profile hidden Markov model," *Pervasive Mob. Comput.*, vol. 72, p. 101336, 2021, doi: 10.1016/j.pmcj.2021.101336.
- [43] J. Spaulding, A. Krauss and A. Srinivasan, "Exploring an open WiFi detection vulnerability as a malware attack vector on iOS devices," *2012 7th International Conference on Malicious and Unwanted Software*, 2012, pp. 87–93, doi: 10.1109/MALWARE.2012.6461013.
- [44] I. Muslukhov, Y. Boshmaf, C. Kuo, J. Lester and K. Beznosov, "Understanding Users' Requirements for Data Protection in Smartphones," *2012 IEEE 28th International Conference on Data Engineering Workshops*, 2012, pp. 228–235, doi: 10.1109/ICDEW.2012.83.
- [45] A. Mathur, L. M. Podila, K. Kulkarni, Q. Niyaz, and A. Y. Javaid, "NATICUSdroid: A malware detection framework for Android using native and custom permissions," *J. Inf. Secur. Appl.*, vol. 58, no. 1, p. 102696, 2021, doi: 10.1016/j.jisa.2020.102696.
- [46] P. Yan and Z. Yan, "A survey on dynamic mobile malware detection," *Softw. Qual. J.*, vol. 26, no. 3, pp. 891–919, 2018, doi: 10.1007/s11219-017-9368-4.
- [47] T. H. Huang and H. Kao, "R2-D2: ColoR-inspired Convolutional NeuRal Network (CNN)-based Android Malware Detections," in *2018 IEEE Int. Conf. Big Data (Big Data)*, 2018, pp. 2633–2642, doi: 10.1109/BigData.2018.8622324.

- [48] V. Rastogi, Y. Chen, and X. Jiang, "DroidChameleon : Evaluating Android Anti-malware against Transformation Attacks," ASIA CCS '13: Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security, China, May 2013, Pages 329–334, <https://doi.org/10.1145/2484313.2484355> pp. 329–334.
- [49] D. Wang, H. Shu, F. Kang, and W. Bu, "A Malware Similarity Analysis Method Based on Network Control Structure Graph." 2020 IEEE 11th International Conference on Software Engineering and Service Science (ICSESS), Beijing China, 16-18 Oct. 2020, doi: 10.1109/ICSESS49938.2020.9237633.
- [50] P. Yan and Z. Yan, "A survey on dynamic mobile malware detection," *Softw. Qual. J.*, vol. 26, no. 3, pp. 891–919, 2018, doi: 10.1007/s11219-017-9368-4.
- [51] M. Yousefi-azar, L. G. C. Hamey, V. Varadharajan, and S. Chen, "Malytics : A Malware Detection Scheme," *IEEE Access*, vol. 6, pp. 49418–49431, 2018, doi: 10.1109/ACCESS.2018.2864871.
- [52] S. M. Gene and W. Hongyan, "Analyzing and Recognizing Android Malware via," 2017 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, 12-14 Oct. 2017 Changchun, China, 2017, pp. 17-20, doi: 10.1109/CyberC.2017.36.
- [53] Meng, X., An integrated networkbased mobile botnet detection system (Doctoral dissertation, City, Universtiy of London).S. Anwar, M. F. Zolkipli, V. Mezhuyev, and Z. Inayat, "A Smart Framework for Mobile Botnet Detection Using Static Analysis," *KSII T Internet Inf.*, vol. 14, no. 6, pp. 2591–2611, 2020, doi: 10.3837/tiis.2020.06.015.
- [54] M. Ashawa and S. Morris, "Analysis of Android Malware Detection Techniques: A Systematic Review," *Int. J. Cyber-Security Digit. Forensics*, vol. 8, no. 3, pp. 177–187, 2019, doi: 10.17781/p002605.
- [55] S. Y. Yerima, S. Sezer, and G. Mcwilliams, "Analysis of Bayesian Classification based Approaches for Android Malware Detection Analysis of Bayesian Classification based Approaches for Android Malware Detection," *IET Inf. Secur.*, vol. 8, no. 1, pp. 25–36, 2014, doi: 10.1049/iet-ifs.2013.0095.
- [56] M. Yusof, M. M. Saudi, and F. Ridzuan, "A new mobile botnet classification based on permission and API calls," in *Proc. - 2017 7th Int. Conf. Emerg. Secur. Technol. EST 2017*, pp. 122–127, 2017, doi: 10.1109/EST.2017.8090410.
- [57] M. Ashawa and S. Morris, "Host-Based Detection and Analysis of Android Malware: Implication for Privilege Exploitation," *Int. J. Inf. Secur. Res.*, vol. 9, no. 2, pp. 871–880, 2019, doi: 10.20533/ijisr.2042.4639.2019.0100.
- [58] U. T. Austin, "Quantifying and Improving the Efficiency of Hardware-based Mobile Malware Detectors," 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), 15-19 Oct. 2016, Taipei, Taiwan, pp. 1–13, doi: 10.1109/MICRO.2016.7783740.
- [59] K. Ariyapala, H. G. Do, H. N. Anh, W. K. Ng, and M. Conti, "A Host and Network Based Intrusion Detection For Android Smartphones," 2016 30th International Conference on Advanced Information Networking and Applications Workshops (WAINA), 23-25 March, Switzerland, 2016, doi: 10.1109/WAINA.2016.35.
- [60] X. Zhong, S. Member, Y. Fu, L. Yu, and R. Brooks, "Stealthy Malware Traffic – Not as Innocent as It Looks," 2015 10th International Conference on Malicious and Unwanted Software (MALWARE), 20-22 Oct. 2015, Fajardo, PR, USA, pp. 110–116, 2015, doi: 10.1109/MALWARE.2015.7413691.

- [61] R. Barona and E. A. M. Anita, "A Survey on Data Breach Challenges in Cloud Computing Security : Issues and Threats," 2017 International Conference on circuits Power and Computing Technologies [ICCPCT], 20-21 April 2017, Kollam, India, doi: 10.1109/ICCPCT.2017.8074287
- [62] S. Hsiao and D. Kao, "The Static Analysis of WannaCry & Ransomware," 20th International Conference on Advanced Communications Technology(ICAICT), 11-14 Feb. 2018, Chuncheon, Korea, pp. 153–158, doi: 10.23919/ICAICT.2018.8323680.
- [63] E. B. Karbab and M. Debbabi, "ToGather : Automatic Investigation of Android Malware," in Proc. 13th Int. Conf. Availab. Reliab. Secur., Aug. 2018, pp. 1-10, doi: 10.1145/3230833.3230870.
- [64] K. S. Yim, I. Malchev, A. Hsieh, and D. Burke, "Treble : Fast Software Updates by Creating an Equilibrium in an Active Software Ecosystem of Globally Distributed Stakeholders," ACM Trans. Embed. Comput. Syst., vol. 18, no. 5, pp. 1-23, 2019, doi: 10.1145/3358237.
- [65] F. Alswaina and K. Elleithy, "Android malware family classification and analysis: Current status and future directions," Electron., vol. 9, no. 6, pp. 1–20, 2020, doi: 10.3390/electronics9060942.
- [66] C. Uppin and G. George, "Analysis of Android Malware Using Data Replication Features Extracted by Machine Learning Tools," Int. J. Sci. Res. Comput. Sci. Eng. Inf. Technol., vol. 5, no. 5, pp. 193–201, 2019, doi: 10.32628/CSEIT195532.
- [67] A. Talha and I. Alper, "An in-depth analysis of Android malware using hybrid techniques," Digit. Investig., vol. 24, pp. 25–33, 2018, doi: 10.1016/j.diin.2018.01.001.
- [68] F. C. Mead, J. M. Zielinski, L. Watkins, and W. H. Robinson, "A Mobile Two-Way Wireless Covert Timing Channel Suitable for Peer-to-Peer Malware." 2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC), 8-13 Oct. 2017, Montreal, QC, Canada, doi: 10.1109/PIMRC.2017.8292638.
- [69] F. Li, A. Lai, and D. Ddl, "Evidence of Advanced Persistent Threat : A Case Study of Malware for Political Espionage," 2011 6th International Conference on Malicious and Unwanted Software, 18-19 Oct. 2011, Fajardo, PR, USA, doi: 10.1109/MALWARE.2011.6112333
- [70] N. Tavabi, N. Bartley, A. Abeliuk, S. Soni, E. Ferrara, and K. Lerman, "Characterizing Activity on the Deep and Dark Web," Companion Preceedings of the 2019 World Wide Web Conference, 13 - 17 May 2019, San Francisco USA. doi:10.1145/3308560.3316502.
- [71] K. Khanmohammadi, N. Ebrahimi, A. Hamou-lhadj, and R. Khoury, "Empirical study of android repackaged applications," Empir. Softw. Eng., vol. 24, pp. 3587-3629, 2019, doi: 10.1007/s10664-019-09760-3s.
- [72] Kaspersky Security Bulletin : OVERALL STATISTICS FOR 2017," Accessed 12 June 2019.[Online].Available:https://media.kaspersky.com/jp/pdf/pr/Kaspersky_KSB2017_Statistics-PR-1045.pdf.
- [73] K. Liu, S. Xu, G. Xu, D. Sun, and H. Liu, "A Review of Android Malware Detection Approaches Based on Machine Learning," IEEE Access, vol. 8, pp. 124579–124607, 2020, doi: 10.1109/ACCESS.2020.3006143.
- [74] S. Turker and A. B. Can, "AndMFC: Android malware family classification framework," 2019 IEEE 30th Int. Symp. Pers. Indoor Mob. Radio Commun. PIMRC Workshops, 2019, pp. 1-6, doi: 10.1109/PIMRCW.2019.8880840.

- [75] Android, "Android Security & Privacy 2018 Year In Review," Accessed on 23 Mar. 2019. [Online]. Available: https://source.android.com/security/reports/Google_Android_Security_2018_Report_Final.pdf.
- [76] M. Stone. (2019). A Deep Dive into Reversing Android Pre-Installed Apps Securing the System. Accessed on 23 Mar. 2019. [Online]. Available: <https://i.blackhat.com/USA-19/Thursday/us-19-Stone-Securing-The-System-A-Deep-Dive-Into-Reversing-Android-Preinstalled-Apps.pdf>
- [77] R. Samani and G. Davis, "McAfee Mobile Threat Report: Mobile Malware Continues to Increase in Complexity and Scope," McAfee, Santa Clara, CA, USA, McAfee Mobile Threat Report Q1, 2019. Accessed on 23 Mar. 2019. [Online]. Available: <https://www.mcafee.com/enterprise/en-us/assets/reports/rp-mobile-threat-report-2019.pdf>.
- [78] R. Kumar, X. Zhang, W. Wang, R. U. Khan, J. Kumar, and A. Sharif, "A Multimodal Malware Detection Technique for Android IoT Devices Using Various Features," in *IEEE Access*, vol. 7, pp. 64411-64430, 2019, doi: 10.1109/ACCESS.2019.2916886.
- [79] N. Tavabi, N. Bartley, A. Abeliuk, S. Soni, E. Ferrara, and K. Lerman, "Characterizing Activity on the Deep and Dark Web," in *Companion Proc. 2019 World Wide Web Conf.*, May 2019, pp. 206–213, doi: 10.1145/3308560.3316502.
- [80] M. Al-Janabi and A. M. Altamimi, "A comparative analysis of machine learning techniques for classification and detection of malware," in *Proc. - 2020 21st Int. Arab Conf. Inf. Technol. ACIT*, 2020, doi: 10.1109/ACIT50332.2020.9300081.
- [81] H. Cai, N. Meng, B. Ryder and D. Yao, "DroidCat: Effective Android Malware Detection and Categorization via App-Level Profiling," *IEEE Trans. Inf. Forensics Secur.*, vol. 14, no. 6, pp. 1455-1470, June 2019, doi: 10.1109/TIFS.2018.2879302.
- [82] R. Samani and G. Davis, "McAfee Mobile Threat Report: Mobile Malware Continues to Increase in Complexity and Scope," McAfee, Santa Clara, CA, USA, McAfee Mobile Threat Report Q1, 2019. Accessed on 23 Mar. 2019. [Online]. Available: <https://www.mcafee.com/enterprise/en-us/assets/reports/rp-mobile-threat-report-2019.pdf>
- [83] R. Kumar, X. Zhang, W. Wang, R. U. Khan, J. Kumar, and A. Sharif, "A Multimodal Malware Detection Technique for Android IoT Devices Using Various Features," in *IEEE Access*, vol. 7, pp. 64411-64430, 2019, doi: 10.1109/ACCESS.2019.2916886.
- [84] P. J. Taylor, T. Dargahi, A. Dehghantanha, R. M. Parizi, and K. R. Choo, "A systematic literature review of blockchain cyber security," *Digit. Commun. Networks*, vol. 6, no. 2, pp. 147–156, 2020, doi: 10.1016/j.dcan.2019.01.005.
- [85] Deloitte Cyber Threat Intelligence, "COVID-19 Global Cyber risks : Is a major cyberattack looming?," Accessed 3 June, 2020. [Online]. Available: <https://www2.deloitte.com/content/dam/Deloitte/global/Documents/AboutDeloitte/COVID-19/gx-cyber-covid-19-deloitte-global-cyber-covid-executive-briefing-issue-8release-date-6-3-2020-vf.pdf>.
- [86] M. Faou, "TURLA LIGHTNEURON One email away from remote code execution," Eset, 25 May, pp. 1–33, 2019, Accessed on 29 April. 2019. [Online]. Available: <https://www.welivesecurity.com/wp-content/uploads/2019/05/ESET-LightNeuron.pdf>
- [87] R. Brown and A. Rocha, "Entrepreneurial uncertainty during the Covid-19 crisis: Mapping the temporal dynamics of entrepreneurial finance," *J. Bus. Ventur. Insights*, vol. 14, p.174, 2020, doi: 10.1016/j.jbvi.2020.e00174.
- [88] A. Balachandar, S. Abdul Alsowdh, and K. Arumugam, "Design and Development of Future Estimate in Confronting Ransomware," *J. Phys. Conf. Ser.*, vol. 17, no. 7, p. 63, 2021, doi: 10.1088/1742-6596/1717/1/012063.

- [89] L. García and R. J. Rodríguez, "A peek under the hood of iOS malware," Proc. - 2016 11th Int. Conf. Availability, Reliab. Secur. ARES, 31 Aug.-2 Sept. 2016, Salzburg, Austria, pp. 590–598, 2016, doi: 10.1109/ARES.2016.15.
- [90] S. I. Imtiaz, S. ur Rehman, A. R. Javed, Z. Jalil, X. Liu, and W. S. Alnumay, "DeepAMD: Detection and identification of Android malware using high-efficient Deep Artificial Neural Network," Futur. Gener. Comput. Syst., vol. 115, pp. 844–856, 2021, doi: 10.1016/j.future.2020.10.008.
- [91] L. Vokorokos, P. Drienik, O. Fortotira, and J. Hurtuk, "Abusing mobile devices for Denial of Service attacks," IEEE 13th International Symposium on Applied Machine Intelligence and Informatics, 22-24, January 2015 Herl'any, Slovakia pp. 22–25, 2015, doi:10.1109/SAMI.2015.7061886.
- [92] F. Martinelli, F. Mercaldo, V. Nardone, A. Santone, and G. Vaglini, "Simulation Modelling Practice and Theory Model checking and machine learning techniques for HummingBad mobile malware detection and mitigation," Simul. Model. Pract. Theory, vol. 105, p.169, 2020, doi: 10.1016/j.simpat.2020.102169.
- [93] H. Wang, W. Zhang, S. Member, and H. He, "An Evolutionary Study of IoT Malware," IEEE Internet of Things Journal vol. 8, no. 20, pp. 15422–15440, 2021, doi:10.1109/JIOT.2021.3063840.
- [94] D. Schultes, "SequelsK — A Bidirectional Swift-Kotlin-Transpiler," 2021 IEEE/ACM 8th International Conference on Mobile Software Engineering and Systems (MobileSoft), 17-19 May 2021, Madrid, Spain, pp. 73–83, 2021, doi: 10.1109/MobileSoft52590.2021.00017.
- [95] C. Li, D. He, S. Li, S. Zhu, S. Chan, and Y. Cheng, "Android-based Cryptocurrency Wallets: Attacks and Countermeasures," in Proc. - 2020 IEEE Int. Conf. Blockchain, Blockchain 2020, pp. 9–16, 2020, doi: 10.1109/Blockchain50366.2020.00010.
- [96] Y. Sun, Y. Chen, Y. Pan, and L. Wu, "Android Malware Family Classification Based on Deep Learning of Code Images," Int. J. Comput. Sci., vol. 46, no. 4, pp. 524–533, 2019.
- [97] K. K. Tam, "Analysis and Classification of Android Malware," Ph.D. dissertation, Inf. Secur. Dep., Univ. London, UK. 2016.
- [98] A. Cimitile, F. Martinelli, and F. Mercaldo, "Machine Learning Meets iOS Malware : Identifying Malicious Applications on Apple Environment," in Proc. 3rd Int. Conf. Inf. Syst. Secur. Priv., pp. 487–492, 2017, doi: 10.5220/0006217304870492.
- [99] J. Li, L. Sun, Q. Yan, Z. Li, W. Srisa-an and H. Ye, "Significant Permission Identification for Machine-Learning-Based Android Malware Detection." In IEEE Transactions on Industrial Informatics, vol. 14, no. 7, pp. 3216–3225, 2018, doi: 10.1109/TII.2017.2789219.
- [100] N. Xie, X. Wang, W. Wang, and J. Liu, "Fingerprinting Android malware families," Fornt. Comput. Sci., vol. 13, pp. 637–646, 2019, doi: 10.1007/s11704-017-6493-y.
- [101] S. Saeed, N. Z. Jhanjhi, M. Naqvi, M. Humayun, and S. Ahmed, "Ransomware: A framework for security challenges in internet of things," in 2020 2nd Int. Conf. Comput. Inf. Sci. ICCIS 2020, 2020, doi: 10.1109/ICCIS49240.2020.9257660.
- [102] Kilber, N., Kaestle, D. and Wagner, S., 2021. Cybersecurity for Quantum Computing. arXiv preprint arXiv:2110.14701.P. Patidar, and D. Kaushal, "Review of Information Protection on Android Application Devices," Int. J. Sci. Res. Eng. Trends, vol. 6, no. 4, pp. 2751–2758, 2020.
- [103] R. Samani, "McAfee Mobile Threat Report Mobile Malware Is Playing Hide and Steal," McAfee, Santa Clara, CA, USA, McAfee Mobile Threat Report Q1, 2020. Accessed on Aug. 2020. [Online]. Available: <https://www.mcafee.com/content/dam/consumer/en-us/docs/2020-Mobile-Threat-Report.pdf>.

- [104] P. Kocher *et al.*, "Spectre Attacks: Exploiting Speculative Execution," 2019 IEEE Symposium on Security and Privacy (SP), 2019, pp. 1-19, doi: 10.1109/SP.2019.00002.
- [105] F. Mercaldo and A. Santone, "Deep learning for image-based mobile malware detection," J. Comput. Virol. Hacking Tech., vol. 16, no. 2, pp. 157–171, 2020, doi: 10.1007/s11416-019-00346-7.
- [106] M. La Polla, F. Martinelli and D. Sgandurra, "A Survey on Security for Mobile Devices," in IEEE Communications Surveys & Tutorials, vol. 15, no. 1, pp. 446-471, First Quarter 2013, doi: 10.1109/SURV.2012.013012.00028.
- [107] S. Shahriar, S. Das, and S. Hossain, "Security threats in Bluetooth technology," Comput. Secur., vol. 74, pp. 308–322, 2018, doi: 10.1016/j.cose.2017.03.008.
- [108] H. Chiang, "Mobile Malware Behavioral Analysis and Preventive Strategy Using Ontology," 2010 IEEE Second International Conference on Social Computing, 20-22 Aug. 2010, Minneapolis, MN, USA doi: 10.1109/SocialCom.2010.160.
- [109] C. Martin, L. Juil Legand, I. Burge, I. Joseph, N. Gill, Alicia, A. Marcus "Modelling the spread of mobile malware." International Journal of Computer Aided Engineering and Technology, vol. 2, no.1, pp. 3-14, 2010, doi:10.1504/IJCAET.2010.029592.
- [110] S. Pirani, A. Johansen, and A. J. Mustill, "Astrophysics On the inclinations of the Jupiter Trojans," vol. 89, no.9, pp. 1–10, 2019, doi:10.1051/0004-6361/201936600.
- [111] S. Kalpana, "A Survey on Rise of Mobile Malware and Detection Methods," 2017 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS), 17-18 March 2017, Coimbatore, India, doi: 10.1109/ICIIECS.2017.8276158.
- [112] W. Chun and D. Park, "Malicious Code hiding Android APP ' s Distribution and Hacking Attacks and Incident Analysis," 2012 8th International Conference on Information Science and Digital Content Technology (ICIDT2012), 26-28 June 2012, Jeju, pp. 686–689, 2011.
- [113] S. Töyssy and M. Helenius, "About malicious software in smartphones," Journal of Computer Virology, vol. 3, no. 4, pp. 109–119, 2006, doi: 10.1007/s11416-006-0022-0.
- [114] S. peng, S. yu, and A. yang, "Smartphone malware and its propagation modeling :," IEEE communications surveys & tutorials, vol. 16, no. 2, pp. 925–941, 2014.
- [115] Y. L. Ho, "Mobile and Ubiquitous Malware," Proceedings of the 7th International Conference on Advances in Mobile Computing and Multimedia, 14th December 2009, pp. 559–563, doi:10.1145/1821748.1821856.
- [116] K. P. Grammatikakis, I. Koufos, N. Kolokotronis, C. Vassilakis, and S. Shiaeles, "Understanding and Mitigating Banking Trojans : From Zeus to Emotet," 2021 IEEE International Conference on Cyber Security and Resilience (CSR), 26-28 July 2021, Rhodes, Greece, pp. 121–128, 2021, doi:10.1109/CSR51186.2021.9527960.
- [117] H. Chi and A. Florida, "Integrate Mobile Devices into CS Security Education", Proceedings of the 2015 Information Security Curriculum Development Conference October 2015 Article pp. 1–4, doi:10.1145/2885990.2885991.
- [118] J. Malik and R. Kaushal, "CREDROID : ANDROID MALWARE DETECTION BY NETWORK TRAFFIC ANALYSIS," Proceedings of the 1st ACM Workshop on Privacy-Aware Mobile Computing, pp. 28–36, 2016, doi:10.1145/2940343.2940348.
- [119] Z. Chen *et al.*, "Machine learning based mobile malware detection using highly imbalanced network traffic," Inf. Sci. (Ny), vol. 433, pp. 346–364, 2018, doi: 10.1016/j.ins.2017.04.044.
- [120] V. Svajcer, "When Malware Goes Mobile: Causes, Outcomes and Cures." Accessed on 25 June 2020. [Online]. Available: https://www.sophos.com/en-us/medialibrary/Gated%20Assets/white%20papers/Sophos_Malware_Goes_Mobile.pdf.

- [121] N. Xie, X. Wang, and W. Wang, "Fingerprinting Android malware families," *Front. Comput. Sci.*, vol. 13, no. 3, pp. 637–646, 2019, doi:10.1007/s11704-017-6493-y.
- [122] C. A. Profiling, "DroidCat: Effective Android Malware Detection and," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 6, pp. 1455–1470, 2019, doi: 10.1109/TIFS.2018.2879302.
- [123] J. Chen, C. Wang, Z. Zhao, K. Chen, R. Du and G. -J. Ahn, "Uncovering the Face of Android Ransomware: Characterization and Real-Time Detection," in *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 5, pp. 1286-1300, May 2018, doi: 10.1109/TIFS.2017.2787905.
- [124] I. A. Chesti, M. Humayun, N. U. Sama and N. Jhanjhi, "Evolution, Mitigation, and Prevention of Ransomware," 2020 2nd International Conference on Computer and Information Sciences (ICCIS), 2020, pp. 1-6, doi: 10.1109/ICCIS49240.2020.9257708.
- [125] T. M. Chen and Y. Rahulamathavan, "PIndroid: A novel Android malware detection," *Comput. Secur.*, vol. 68, pp. 36–46, 2017, doi: 10.1016/j.cose.2017.03.011.
- [126] S. Hojjatinia, S. Hamzenejadi and H. Mohseni, "Android Botnet Detection using Convolutional Neural Networks," 2020 28th Iranian Conference on Electrical Engineering (ICEE), 2020, pp. 1-6, doi: 10.1109/ICEE50131.2020.9260674.
- [127] F. Pierazzi and C. London, "A Data-driven Characterization of Modern Android Spyware," *ACM Transactions on Management Information Systems*, vol. 11 no. 1, pp. 1-38, 2020, doi:10.1145/3382158.
- [128] Z. H. U. Yixiang, "Review of iOS Malware Analysis," 2017 IEEE Second International Conference on Data Science in Cyberspace (DSC), 26-29 June 2017, Shenzhen, China, pp. 4–8, 2017, doi: 10.1109/DSC.2017.104.
- [129] Y. Chen, "Dynamic Android Malware Analysis with De-Identification of Personal Identifiable Information," 2020 the 3rd International Conference on Computing and Big Data, August 2020, Pages 30–36, doi: 10.1145/3418688.3418694.
- [130] D. Emm, R. Unuchek, M. Garnaeva, A. Ivanov, D. Makrushin, and F. Sinitsyn, "IT THREAT EVOLUTION IN Q2 2016," Kaspersky lab report. Accessed 2nd July 2021. [Online]. Available: https://media.kasperskycontenthub.com/wp-content/uploads/sites/43/2018/03/07185743/Kaspersky_Q2_malware_report_ENG.pdf.
- [131] F. Mercaldo and A. Santone, "Deep learning for image-based mobile malware detection," *J. Comput. Virol. Hacking Tech.*, vol. 16, no. 2, pp. 157–171, 2020, doi: 10.1007/s11416-019-00346-7.
- [132] F. Wei, Y. Li, S. Roy, X. Ou, W. Zhou, "Deep Ground Truth Analysis of Current Android Malware. In: Polychronakis M., Meier M. (eds) Detection of Intrusions and Malware, and Vulnerability Assessment. DIMVA, vol 10327, 2017, Springer, Cham, doi:10.1007/978-3-319-60876-1_12.
- [133] S. Garg and N. Baliyan, "Comparative analysis of Android and iOS from security viewpoint," *Comput. Sci. Rev.*, vol. 40, p. 100372, 2021, doi: 10.1016/j.cosrev.2021.100372.
- [134] A. Mujawib, S. Duraibi, J. Song, "IoT-Taint: IoT malware detection framework using dynamic taint analysis, "2019 International Conference on Computational Science and Computational Intelligence (CSCI)," 5-7 Dec. 2019, Las Vegas, NV, USA, pp. 2019–2022, doi: 10.1109/CSCI49370.2019.00229.
- [135] S. Garg and N. Baliyan, "Comparative analysis of Android and iOS from security viewpoint," *Comput. Sci. Rev.*, vol. 40, p. 100372, 2021, doi: 10.1016/j.cosrev.2021.100372.

- [136] P. Wu, D. Liu, J. Wang, B. Yuan, and W. Kuang, "Detection of Fake IoT App Based on Multidimensional Similarity," vol. 7, no. 8, pp. 7021–7031, 2020.
- [137] H. Talal and R. Zagrouba, "A Robust Framework for MADS Based on DL Techniques on the IoT," *Electronics*, vol. 10, no. 21, p. 2723, Nov. 2021.
- [138] A. Josias and S. Runturambi, "Analyzing Cyber Crimes during COVID-19 Time in Indonesia," 2021 3rd International Conference on Computer Communication and the Internet, 25-27 June 2021, Nagoya, Japan, pp. 78–83, 2021, <https://doi.org/10.1109/ICCCI51764.2021.9486775>.
- [139] K. Khariwal, J. Singh and A. Arora, "IPDroid: Android Malware Detection using Intents and Permissions," 2020 Fourth World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4), 2020, pp. 197-202, doi: 10.1109/WorldS450073.2020.9210414.
- [140] A. Talha and I. Alper, "An in-depth analysis of Android malware using hybrid techniques," *Digit. Investig.*, vol. 24, pp. 25–33, 2018, doi: 10.1016/j.diin.2018.01.001.

CHAPTER 3: ANALYSIS OF ANDROID MALWARE DETECTION TECHNIQUES: A SYSTEMATIC REVIEW

The content of this chapter has been published in the International Journal of Cyber-Security and Digital Forensics (IJCSDF).

Citation: M. Ashawa and S. Morris, “Analysis of Android Malware Detection Techniques: A Systematic Review,” *Int. J. Cyber-Security Digit. Forensics*, vol. 8, no. 3, pp. 177–187, 2019, doi: <http://dx.doi.org/10.17781/P002605>.

This chapter is linked to the second objective of the research. After extensive review, we identified the research need to investigate the existing techniques based on their approach to identify their strengths and weaknesses. This objective is significant because it identified the limitations and strengths in existing techniques that need improvement for better detection. Identifying some of the limitations and strengths in the existing models was leveraged on building a robust framework and techniques that detect and classify features of sophisticated and emerging mobile malware. The identification of the strengths of the technique was leveraged to carry out a preliminary study in **chapter 5 (published in paper 3)** to determine the feasibility of using them in the main framework.

Abstract

The emergence and rapid development in complexity and popularity of Android mobile phones have created proportionate destructive effects from the world of cyber-attack. Android-based device platform is experiencing great threats from different attack angles such as DoS, Botnets, phishing, social engineering, malware, and others. Among these threats, malware attacks on android phones have become a daily occurrence. This is because Android has millions of users, high computational abilities, popularity, and other essential attributes. These factors influence cybercriminals (especially malware writers) to focus on Android for financial gain, political interest, and revenge. This calls for effective techniques that could detect these malicious applications on android devices. This paper aims to provide a systematic review of the malware detection techniques used for android devices. The results show that most detection techniques are not very effective to detect zero-day malware and other variants that deploy obfuscation to evade detection. The critical appraisal of the study identified some of the limitations in the detection techniques that need improvement for better detection.

Keywords: Malware, detection techniques, Android mobile device, detection rate, static detection, hybrid detection, dynamic detection, cybersecurity.

3.1 Introduction

Android has become the most popular operating system in the world of mobile telephony with the largest users in different parts of the world. A vast number of financial applications such as mobile banking and online banking run on this most popular mobile OS. Other sensitive information like health records, passwords, and usernames are stored on android phones. The technological progression in Android has created proportionate attraction by malware writers who are advancing daily to gain financial roots by creating malware applications that can directly break into android mobile device security and snip victims' data or ransom requests usually inform of bitcoin after a successful attack is implemented. Malware attack on android has generated critical issues in the mobile and security industry at large. These factors increased the occurrence of mobile malware, especially on the Android platform. This research provides a systematic review by formulating and answering research questions using a methodology provided in section 3. This helps to identify relevant works that are carried out on android malware detection techniques with their strengths and limitations.

3.1.1 Malware

Different definitions of malware are given by different scholars and researchers depending on the attack vector deployed or the harm caused. Though these definitions may be different, all converged to the same meaning as malicious applications with evil intent. The research of [1-3] defined malware as simply malicious codes. This definition considers malware as any piece of program segment which is developed with harmful intentions. This definition does not encompass many attacks and harm trajectories followed by malware. Looking at information harvest and data leakage, the study of [4-6] defined malware as any computer program that leaks users' private data without their consent. The research cogitates malware as programs that do not necessarily cause harm to a device but gather information about the activities of such a device and thereafter create disclosure of them to third parties for future attacks. The literature survey on the SOA project [7] looked at malware as any suspicious program that affects organizational databases.

Kaspersky [8] looks at malware as a developed computer program that contaminates and exacts damage on users' computers. As wonderful as the above definitions might be, their coverage is limited when looking at the broad spectrum of malware attacks and effects. This research, therefore, defines malware as any computer code written with evil motives to get unauthorized access into IT infrastructures and digital devices by breaking their security defence parameters and exploiting their vulnerabilities leading to information harvest, data loss, information leakage, file infections, buffer overflow, interruption of computational operations and leading to subsequent physical or operational damage or both.

3.1.2 Malware attacks on Android phones

Malicious programs that are designed to attack android Dalvik virtual machine (DVM) and java core libraries respectively can be described as Android malware. The majority of Android users download free applications without critically considering whether such applications are genuinely provided by Google or not. Many do not turn on their Android device application permission monitor [9] to help confirm Apps checked and guaranteed by Google bouncer [10] before installation even when such application over request permission to different resources of the device. The study of [11] asserted that application download from unknown sources constitutes one of the major attack vectors through which mobile devices get infected with malware. Besides, vigilance, when given permission privileges to applications with, is a good security practice. Some

Android malicious applications exploit a vulnerability in the system when such permissions are granted at the installation stage after the application download.

Evasion and obfuscation techniques deployed by malware to elude detection have made Google play store insecure despite the tremendous effort of Google and the associated companies to review Android applications to avoid malware distribution. Android malware keeps emerging daily. On August 7th, 2018, a security network called Palo Alto [12] again identified about 145 malicious applications on the Google play store which appear to be genuine but are not. There are a lot of detection techniques used by many researchers and security companies to detect and prevent this threat (malware) but none of these seems to be perfect.

For knowledge contribution, this study will provide general knowledge on mobile malware and some of the attack tactics malware deploy to execute their payloads and exploit Android vulnerabilities. This knowledge will help to develop further security parameters and tools that will improve Android mobile security. Evaluation of the detection techniques will give an idea on which is more efficacious in detecting malware on android platforms and which needs to be improved for better detection of sophisticated malware variants. The study provides some of the challenges that digital forensics and cybersecurity investigators may encounter when working on Android malware. The remaining part of this paper is organized into the following sections: Section 3.2 provides the related work. The methodological approach to the research is provided in section 3.3. Sections 3.4, 3.5, 3.6, and 3.7 are for detection technique, discussion, evaluation, and conclusions respectively.

3.2 Related work

Research in Mobile security has become a thing of concern. A range of research in mobile technologies from design, vulnerability, threats, and detection techniques is ongoing. Many security industries are spending billions of funds on this field. This paper examined some of them as building blocks. The study of [13] x-rayed the dynamic detection and analysis approach on how malware can be detected on the Gingerbread Android version. DroidScope detected DroidKungFu and DroidDream android malware were the major malware investigated in this research. Detection features obtained by the research indicated that while DroidDream encrypted IMSI and IMEI numbers into XML String, DroidKungFu carried cage exploit to gain unpermitted access to the mobile device security. The outcome of the study demonstrated the efficiency of the technique

used was resistant to malware code obfuscation. The observed limitation was that the research did not consider analysing mobile RAM for forensics artefacts after executing the malware. In addition, fundamental features such as core logic, exploit binaries and native libraries were failed to be analysed by the technique approach.

The research of [14] made a comparative analysis of static, dynamic, and machine learning detection techniques used for malware detection on android applications. Market Centralization in mobile platform applications has made malware detection a very tedious task for most detection techniques even for machine learning and AI techniques. For instance, Google plays the central market role for Android Google bouncer to verify the legality of many apps. This security monitoring is insufficient because millions of android developers associated with Google who's their Apps are not properly scrutinized before being permitted on the play store. This is similar to the phone store and App store for Windows and Apple respectively. This made the security of mobile devices especially Android-based not to be 100% absolute. The detection approach considered was based mainly on misuse and anomaly with the application, untrusted data, and system state as the main objectives of the analysis. None of the techniques provided a 100% detection rate for mobile malware, based on the result obtained.

Anusha [15] dynamically compared malicious and mobile application behaviour using mobile API for malware detection. The developed detection system (WMMD) detected obfuscated mobile malware which antivirus software failed to detect. The detection approach used Finite State Automata (FSA) for malware code sampling and pattern checking. Using analysis by run-time, the model detected viruses both before and after packing. Of all the six (6) antivirus software used, none was able to detect those mobile viruses after packing with UPX. It is then perceived that most anti-virus programs are not reactive to malware detection because they are signature-based. This result was similar to [16] which is based on behavioural analysis for detecting malware on Android applications. The study mined 216 and 278 for normal and malicious Android applications separately. A variety of trained mathematical algorithms were applied to both the benign and malicious data set. Using correlational analysis, the research realized 97.16% detection accurateness.

Looking at advanced malware artefacts in the mobile memory [17] performed an analysis of malware detection in mobile memory using memory forensics methodology. The research detected with 90% accuracy a self-replicating Trojan with 20% unclassified samples. The research observed that significant malware information can be mined from memory dump analysis of android mobile devices. Hidden codes waiting to explode at favourable conditions are easily exposed. The research however could not provide any investigation on searching malware attributes that are significant for forensic and security analysis. There have been elaborate works on detection techniques but there has not been research that identifies and lists the limitations and strengths that existed in those techniques. By identifying both the limitations and the strengths will help improve the efficiency of those techniques and enhance better detection of malware on Android devices.

3.3 Methodology

An organized appraisal concerning the comprehensive malware detection framework outlined by [13] was applied in this paper. The analysis targets to investigate recent studies on android malware detection techniques. The subsequent methodological subheadings expound on the research questions, criteria for paper selection, source of data, and research keywords, a summary of the research interpretation were presented.

3.3.1 Research questions (RQs)

Based on the review in section two to identify potential research gaps, the following research questions were formulated: What are the techniques used for detecting malware attacks on android devices? What are the key limitations when that exist when using those techniques? Are there some areas of strength that are found in those techniques?

3.3.2 Criteria for papers selection

In selecting papers for this study, some standards were set. Only papers that matched and met these criteria were selected. The selection criteria are as follows:

- The paper must be written in the English Language.
- The paper must be published within three years (2016 to 2018) apart from frameworks

- It must discuss android malware and detection techniques using “OR” or “AND” operators. The operators here mean a paper can discuss one or both phrases.
- The selected paper must reflect the research experimental evidence on android malware detection in its content and results.

3.3.3 Justification for the selection criteria

English was selected as the language for the study to allow wide coverage for readers. Papers written in native languages are limited to a particular tribe of people with little impact. The choice to go for three years' publications is to produce research with the current trend in malware and their detection techniques paradigms. Frameworks could be from any year since they form bedrocks for any study. The operators “OR” and “AND” means that the selected paper must discuss either android malware, mobile malware detection techniques, or both. Finally, the paper must show the experimental results with the approach used for such a detection technique.

3.3.4 Source of data

To get reliable data sources, reputable academic research databases encompassing computational disciplines with high publication reputations. Such databases included but were not limited to IEEE, Springer, ACM, Wiley, and Inderscience. Conference papers and journals from those sources formed the primary source of data for the study. Papers from untrusted sources such as Wikipedia were not considered since they are rated unreliable. However, data could be sourced from dependable blogs such as SAN blogs because of their reputation. As stated in the paper selection criteria above, basic search operators which include ‘OR’ and ‘AND’s will be enforced to get request data within the defined research objectives. The main data which we intend to mine are the technical challenges, limitations, strengths of detection techniques used for android malware. Experimental results and the approach deployed by each research will not also be discarded.

3.4 Android malware detection techniques

For this survey, the following malware detection and classification techniques were examined:

3.4.1 Dynamic detection techniques

Android-based applications correlate with the device OS via system calls which makes it possible to monitor what transpires between them. Dynamic detection technique monitors android malware in a controlled environment at runtime by taking cognizance of the malware pointers which detection signatures can be modelled using them. It inspects malware interaction with mobile resources and services such as location, network, package, OS activities. For the safety of the experimental equipment (physical device), it is recommended that the code execution be carried out in a cybernetic environment.

The study of [18] applied this detection technique on 4034 and 10024 malware and benign datasets respectively. Using the ServiceMonitor approach, the random forest classification algorithm was detected with 96% accuracy of malware on those applications. Using k-fold validation and Markov chain, the classifier module was coached to extract the sample features. Information retrieved such as phone IMEI by the malware was detected to be 67% accurate. Among the detected malware, 17% of the applications were observed to have attached their payload on the device for premium service rating. The mobile utilities such as CPU and Memory were observed to be infected with an overhead device performance of 0.8% and 2% respectively.

Some malware remains dormant on the device after download and installation until an action is triggered. While some do so, others execute their payload at download, installation, and runtime. Access authorization habitually permitted by Android users during application download and installation creates a large space in the device attack vector even though default permissions are always encountered during download and installation sessions. Malicious code attaches benign applications during that exercise. Critical monitoring at these stages is required for the better security of mobile platforms.

3.4.2 Static detection techniques

Static detection technique for malware detection does not execute or run the malware code but solely depends on the malware abstraction features. For malware detection using this technique, the dependable features for detection come from the application byte code or its manifest file; unlike the dynamic method which focuses on the system calls and application program padding. Android applications are in APK format or archive, usually in a zip package. All the Android files, folders, and other resources are included. For meaning detection, reverse engineering is mostly applied to the APK files for features mining. When looking for relevant feature extraction, the manifest file “AndroidManifest.xml” is first to be considered. This manifest file contains permission vector features for access to the installation, locations, battery optimization, and phone state permissions.

Ankita [19] used 103 and 97 malware and benign applications dataset respectively and detected malware on Nexus 5 with API level 19 detected high unauthorized permission attacks by malware. Reverse engineering was used as the experimental approach while applying Naïve Bayes, simple logic, RF 100, RF 10, J.48, Sequential minimal optimization, and IBK algorithms. The XML parser extracted the permission request which generated binary features of the malware which was stored in Attribute Relation File Format (ARFF). Result provided a 96.6% detection rate when the random forest algorithm was used with a 0.069% marginal difference from the worst detection algorithm.

Malicious applications appearance cannot be easily seen until the code running the application is thoroughly inspected using trained systems. To analyse the raw data, which is been processed as a Dalvik bytecode, concatenation of the opcode by disassembling the APK files constitutes good practice. De-compilation of this file provides convolutional direction for extracting and analysing further Android applications files such as XML and other resource files. This approach is similar when using the n-gram procedure in detecting malware. To ascertain and validate experimental results obtained in terms of detection accuracy, the study used four different detection algorithms to detect android malware with a large dataset of 5,560 malware samples. Bytecodes dichotomize CFGs from the object node at initialization. Different detection obtained by the trained algorithms characterized the power of DSA when applied at the input and extraction layer of the model. The random forest algorithm obtained a detection accuracy of 97% higher than the rest.

3.4.3 Hybrid detection techniques

This technique combines both the features of dynamic and static techniques to provide a more robust detection result when analysing malware. The hybrid detection approach to malware detection involves training and detection phases which could be done by dynamic and static techniques respectively. This seems to give a better detection rate than dynamic and static techniques since the strengths in both methods are synergised. Using the deep learning aspect of artificial intelligence, [20] developed and trained a DroidDetector model with some algorithms for android malware detection. The hybrid technique collected a total of 192 android malware and benign samples for training. The model yielded detection result accuracy of 96.60% with 0.0021% disparity amongst the algorithms used.

In some complicated cases where the malware sample is unknown, training and detection may not be done simultaneously to avoid features meddling. Hidden Markov Models prove to have high-performance features when it comes to bisectional improvement in malware detection. The hybrid technique helps to make a comparative analysis of static and dynamic detection rate accuracy. Through the semantic approach of this technique, the study of [21] performed opcode and API call malware sample sequences were extracted using Hidden Markov Models. Recall, precision and specificity determined the threshold of the ROC curve. To define and establish the maliciousness and benignity of an application, Android Buster Sandbox was used as an analyser. Android malware detection by applying API call sequence could not however overcome the problem of malware obfuscation. In addition, the observational sequence of the malware features does not produce a relational correspondence to the HMMs distinct states, this approach cannot generate the initial malware distribution state in the call graph and sequence respectively.

Similarly, the research of [22] used API call graphs to extract malware smali files. Out of the 1,216 suspicious Android applications, a total of 1022 extraction was made. The resultant detection accuracy was found to be 96.12%. Though evasion attacks in Android malware were overcome by this approach, android poisoning attacks could not be addressed using Machine learning. Focusing on API block calls, the study of [23] designed a detective tool (Droiddelver) with a Deep Belief Network algorithm that mined asemantic traces of both known and unknown malware. Boltzmann generated a restricted bipartite graph at the model input layer during malware probabilistic distribution. The malware print tack most at times in such a scenario provides indices of the smali program the malware might be intending to carve on the Android mobile kernel.

The unzipping and decompiling of Android applications before extraction the API call layer requires a smali code to stand between the Dalvik VM and the App interface. Some Android malware is designed to harvest information related to system calls, file systems, mobile location, and images captured by the device camera. Malicious app with this target makes the user of the device physically and informationally vulnerable. He can be easily tracked down and attacked or his system files could be exploited for financial gain or otherwise. This was demonstrated in the study of [24] with a relatively small Android malware dataset.

3.4.4 Permission-based detection techniques

This technique involves a detailed analysis of all the network traffic packets coming from the HTTP server. The analysis of such packets indicates the nature of data that an application or device is sending to or receiving from an isolated server. Some categories of mobile malware do not execute visible and noticeable harm to the host device but only leak PII information such as a list of apps, address, photos, IMEI and IMSI, location and mac address to malicious URL especially when the insecure channel of communication is used. When these traffics are captured by software like Wireshark and are analysed, information leaked might be obtained. It is observed that when a proper approach is deployed, sniffed data by such malware could be detected. When analysing malware with this method, features such as communication protocol and APK files should be the target. In addition, a check can be performed to see if PII data is involved in the captured traffic. However, some Android malware does not generate network traffic to the HTTP URL. This then becomes very difficult to detect malware conducting premium rates to contact numbers. This research is similar to the research of [25-27] respectively.

3.4.5 Emulation-based detection

This technique requires providing a simulated ecosystem by an emulator for running malware samples to separate them from the actual physical resources of the device. Simulation can be done on the Android OS or hardware. However, detection becomes much tougher when the execution of malware is done in the mobile real OS. This technique requires the building of sandboxes and configuring virtual machines systematically and securely to avoid infecting other devices on the network. This technique is effective especially when the Dalvik file (.dex) [28] is properly monitored. Malicious applications can be detected in the sandbox system by obtaining the dex file

and converting it into a form that can be understood by humans. Zero-day malware [29] and malware that escalate privileges [30] are effectively captured with this technique. However, some malware becomes aware of the virtual nature of the environment and tends to evade detection. Different machine techniques are applied to detect malware, network intrusions, and other threats that keep emerging daily (see chapter 4 for details).

3.5 Discussion

From the comparative analysis of this study, basic observations were made regarding the detection techniques. The use of a small malware dataset was one of the basic limitations observed in the investigated techniques. This hinders the true evaluation of the detection efficiency since the sample size could not cut across different edges of malware families. With such a sample size, the technique might have seemed to perform proficiently but when implemented on a larger dataset, the opposite of the result is the case. This could produce a lopsided ratio with little optimization.

Another logical observation was the execution of malware in the sandbox without disabling the Android supervisory calls. This is clear that some malware would likely evade detection by detecting the presence of a sandbox environment by comparing different pieces of information from the system with strings such as “VMware” or “QEMU.” Malware families or samples with the ability to test `sys_vendor` files will be able to detect the sandbox analysis environment especially when the analysis is performed with root privileges. According to the research of [31-32], some of the malware can detect chroot by matching `/proc/1/mountinfo` with the PID of the malicious application information. This can be seen in the case when a known Linux malware known as Handofthief tried to evade IBM virtual machinery. When a virtual environment is perceived, dangerous malware can delay its execution while in the environment and wait at a suitable time to resurface, thus escaping detection.

It, therefore, means that some of the detection techniques whose detection rate amounted to over 90% without disabling the Android supervisory call during malware analysis in the sandbox might be questionable. Variation in detection rate by the same algorithm in different detection scenarios is worrisome. For instance, in the study of [33], a Naïve algorithm was implemented for detection using the CFG approach. The detection rate was found to be 87.0%. In the same manner, 67.64% detection accuracy was witnessed in [34] when CFG was used. It is significant investigating into this wide variation.

3.6 Evaluation

Analysis of android malware detection techniques is significant to building an efficient detection tool by applying both the strengths and limitations identified in all the studied approaches. The static detection methodology mined android metadata and other artefacts from Android malicious applications. Malicious interaction with the dex files at the Dalvik layer provides a dynamic approach to detection. A combination of static and dynamic techniques constitutes a hybrid method that provides a better detection accuracy. Other techniques studied are content and emulation-based detection. Worthy to note in this study is fact that both dynamic and static techniques can be approached in different ways by applying a diverse set of trained algorithms. The dynamic technique can overcome strings of detection issues such as malware fitting. The observed limitation is its susceptibility to transformation attacks, vulnerability to mimicry attacks, and its inability to run on unrooted Android devices.

When opcode sequence approached overcomes the need for hand-engineering. This could not however address the problem of malware encryption. Malware Obfuscation to escape detection is tranquil when this approach is used. Some elements of human interference could lead to unreliable outcomes by using CFG. CFG is Susceptible to malware loading and replication, though highly scalable. The content-based approach provides speedy execution of large malware datasets. This approach achieves virtually no tangible detection result when the APK file is not generating network traffic to HTTP, TCP, and UDP servers. As observed by [35-36], several challenges are confronting android mobile forensic including malware detection.

The detection challenges range from known to unknown, simple to sophistication. Transformation of the designed behavioural model could lead to malware obfuscation when the trained algorithm(s) and mutation approach are known by hackers or malware writers. Mutation and obfuscation make detection very difficult. Malware sandboxing is observed to be a delicate exercise. Little mistake to put the physical device at risk. From this research, it is clear that no detection technique developed and used by industries and individuals is 100% efficient in malware detection. As a result, the occurrence of android malware has become a daily attack threat to the users. The comparative analysis of each detection technique studied in this research is summarized in Table 3-1.

Table 3-1 Comparative analysis of the studied works

S/No	Research	Detection technique	Detection Approach	Algorithms	Detection accuracy	Strength	limitations
37	(Salehi and Amini, 2017)	Dynamic	ServiceMonitor	Random forest, Markov chain	86%	Overcame problem of fitting	Susceptible to transformation and mimicry attacks
38	Ravula et al., 2013)	Static	Reverse engineering	Naïve Bayes, simple logic	96.6%	Overcomes issues of Bytecode Encryption	Fail to execute using Monkey Runner
4	(Yuan et al, 2016)	Hybrid	AI, Deep learning DroidDetector	Multi-layer perceptron, Naïve Bayes and Logistic regression	94.60%	High-level learning representation	Lopsided ratio, Little optimization
39	(Damodaran et al, 2017)	Hybrid	Hidden Markov Models (HMMs)	N/A	N/A	Known and unknown malware samples were detection	The problem of imbalance and obfuscation
22	(Hou et al., 2016)	Hybrid	Deep learning framework	Neural Network	92.66 %	Malware image recognition	Malware depth features were not extracted, assembly language is required
40	(Leeds et al., 2016)	Hybrid	Permissions data flow	Machine learning algorithm	80%	N/A	The sample was not streamlined
41	(McLaughlin et al, 2017)	Static	Opcode sequence	Convolutional neural network (CNN)	87%	The need for hand-engineered was removed	This could not address the problem of malware encryption

42	(Jia et al, 2017)	static	CFG and Bigram using DSA	Random Forest, Naïve Bayes, AdaBoost, Linear SVM	87.0%	Efficiency and scalability can be achieved with this approach	Susceptible to malware loading and replication
43	(Malik and Kaushal, 2016)	Content-based	CREDROID and Web of Trust	N/A	63%	Fast execution	Fails when APK is not generating network traffic.
44	Alimardani and Nazeh, 2019)	Permission-based	Machine learning	LSTM Decision tree Nearest neighbour AdaBoost	LSTM 91% DNN 82.17%	Detects Zeroday, privilege escalation malware	Cannot determine how malware processes the data affected, detected virtual environment
45	(Zhu et al., 2017)	Static	Support vector machine	Random forest	89.9%	Very fast and cost-effective	Bias and variance in features detection

3.7 Conclusion

In this paper, a comparative examination of different Android mobile malware detection techniques was presented. The study was able to identify each of the limitations and strengths in each of the studies' detection techniques through a critical evaluation procedure. The results obtained from this study reinforce the assertion that detection approaches designed for Android malware do not produce 100% efficient detection accuracy. This segment of the research presents a critical evaluation of the reviewed papers. The rationale behind making this comparative analysis is to give a well-defined understanding of the strengths and weaknesses that were identified during the study in the selected detection techniques. A comparative survey on detection techniques focusing primarily on identifying Android malware detection techniques with their respective detection approaches, detection accuracy, and their corresponding strengths and limitations has not been explored before. For further research, we intend to carry out a study on how to provide a security perimeter defence around Google bouncer for efficient Android applications review from third parties before uploading to the play store.

3.8 Future work

There are many prospects and opportunities to further the work presented in this research as some of the future gaps have been identified. First, hybrid and dynamic detection frameworks can be improved with better detection simulation using AI techniques and deep learning tools rather than just applying machine learning algorithms. As identified in section 3.6 (see Table 3-1), an improvement in hybrid detection solutions can help increase efficiency in code coverage and sample streamline. Furthermore, the integration of hybrid detection emulators and physical Android phones will help solve the problem of VM-ware detection and evasion by sophisticated malware such as polymorphic malware that can detect the virtual environment. This future gap when closed will improve the accuracy of this solution. Finally, further research should be carried out on investigating and evaluating detection parameters optimization when comparing Android detection techniques. This should start by carrying out preliminary investigations on detection techniques and rules for host-based malware.

Acknowledgement

This project was fully funded by the Petroleum Technology Development Fund (PTDF) of Nigeria (No. PTDF/ED/PHD/AMA/1245/17)

REFERENCES

- [1] L. Li et al., "Understanding Android App Piggybacking: A Systematic Study of Malicious Code Grafting," in *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 6, pp. 1269-1284, June 2017, doi: 10.1109/TIFS.2017.2656460.
- [2] A. Fuad and A. Rahman, "Securing Sensor to Cloud Ecosystem using Internet of Things (IoT) Security Framework," *ICC '16: Proceedings of the International Conference on Internet of things and Cloud Computing* March 2016 Article No.: 79Pages 1–5<https://doi.org/10.1145/2896387.2906198>.
- [3] C. Yan, F. Zahedi, "Individuals' Internet Security Perceptions and Behaviors: Polycontextual Contrasts Between the United States and China", *Mis Quarterly*, vol. 40, no. 1, pp. 205-222, 2016.
- [4] Z. Wang, C. Li, Z. Yuan, Y. Guan, and Y. Xue, "DroidChain : A novel Android malware detection method based on behavior chains," *Pervasive Mob. Comput.*, vol. 32, pp. 3–14, 2016, doi: 10.1016/j.pmcj.2016.06.018.
- [5] S. Wu, P. Wang, X. Li, and Y. Zhang, "Effective detection of android malware based on the usage of data flow APIs and machine learning," *Inf. Softw. Technol.*, vol. 75, pp. 17–25, 2016, doi: 10.1016/j.infsof.2016.03.004.
- [6] A. Saracino, D. Sgandurra, G. Dini and F. Martinelli, "MADAM: Effective and Efficient Behavior-based Android Malware Detection and Prevention," in *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 1, pp. 83-97, 1 Jan.-Feb. 2018, doi: 10.1109/TDSC.2016.2536605.
- [7] A. Shashwat, D. Kumar and L. Chanana, "An end-to-end security framework for service oriented architecture," *2017 International Conference on Infocom Technologies and Unmanned Systems (Trends and Future Directions) (ICTUS)*, 2017, pp. 475-480, doi: 10.1109/ICTUS.2017.8286056.
- [8] A. Gostev, "Kaspersky Security Bulletin Malware evolution 2008," pp. 1–28, 2008. [Online]. Accessed, 11 Nov. 2019. Availability: https://media.kasperskycontenthub.com/wp-content/uploads/sites/43/2018/03/20140350/kaspersky_security_bulletin_part_1_threats_en.pdf.
- [9] A. V. P. N. P. Apps, M. Ikram, N. Vallina-rodriguez, S. Seneviratne, and U. C. Berkeley, "An Analysis of the Privacy and Security Risks of Android VPN permission-enabled Apps," pp. 349–364, 2016. Doi: <https://doi.org/10.1145/2987443.2987471>.
- [10] W. Diao, "Evading Android Runtime Analysis Through Detecting Programmed Interactions," *WiSec '16: Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks* July, pp.159–164, 2016, doi: <https://doi.org/10.1145/2939918.2939926>.
- [11] Y. Tanaka, M. Akiyama, and A. Goto, "Analysis of malware download sites by focusing on time series variation of malware," *J. Comput. Sci.*, vol. 22, pp. 301–313, 2017, doi: 10.1016/j.jocs.2017.05.027.
- [12] A. Davies, "Android malware infection", [Online]. Accessed, Jan. 2019. Availability: <https://www.lincolnshirelive.co.uk/news/local-news/check-your-apps-now-145-1876747>.
- [13] L. K. Yan, "DroidScope : Seamlessly Reconstructing the OS and Dalvik Semantic Views for Dynamic Android Malware Analysis." In: *Symp. 2017. USENIX security symposium*, pp. 569—584, 2017.
- [14] A. Skovoroda and D. Gamayunov, "Review of the Mobile Malware Detection Approaches," *2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, 2015, pp. 600-603, doi: 10.1109/PDP.2015.54.
- [15] A. Damodaran, F. Di Troia, C. A. Visaggio, T. H. Austin, and M. Stamp, "A comparison of static, dynamic, and hybrid analysis for malware detection," *J. Comput. Virol. Hacking Tech.*, vol. 13, no. 1, pp. 1–12, 2017, doi: 10.1007/s11416-015-0261-z.

- [16] S. Dai, Y. Liu, T. Wang, T. Wei and W. Zou, "Behavior-Based Malware Detection on Mobile Phone," 2010 6th International Conference on Wireless Communications Networking and Mobile Computing (WiCOM), 2010, pp. 1-4, doi: 10.1109/WICOM.2010.5601291.
- [17] L. Singh and M. Hofmann, "Dynamic behavior analysis of android applications for malware detection," 2017 International Conference on Intelligent Communication and Computational Techniques (ICCT), 2017, pp. 1-7, doi: 10.1109/INTELCCT.2017.8324010.
- [18] Majid, S., Amini, M.: Android Malware Detection using Markov Chain Model of Application Behaviors in Requesting System Services." 2017, doi: arXiv preprint arXiv:1711.05731.
- [19] K. Ankita, F. D. Troia, M. Stamp, "Static and Dynamic Analysis of Android Malware." In ICISSP, pp. 653-662, 2017.
- [20] Z. Yuan, Y. Lu and Y. Xue, "Droiddetector: android malware characterization and detection using deep learning," in Tsinghua Science and Technology, vol. 21, no. 1, pp. 114-123, Feb. 2016, doi: 10.1109/TST.2016.7399288.
- [21] M. Stamp, D. F. Anusha, "A comparison of static, dynamic, and hybrid analysis for malware detection," Journal of Computer Virology and Hacking Techniques, vo. 13, no. 1, pp. 1-12 2017.
- [22] L. Chen, S. Hou, Y. Y. B, and L. Chen, "An Adversarial Machine Learning Model Against Android Malware Evasion Attacks," vol. 1, pp. 43–55, 2017, doi: 10.1007/978-3-319-69781-9.
- [23] S. Hou, A. Saas, Y. Y. B, and L. Chen, "DroidDelver : An Android Malware Detection System Using Deep Belief Network Based on API Call Blocks," vol. 1, pp. 54–66, 2016, doi: 10.1007/978-3-319-47121-1.
- [24] M. Leeds, M. Keffeler, and T. Atkison, "A comparison of features for android malware detection," Proc. SouthEast Conf. ACMSE 2017, pp. 63–68, 2017, doi: 10.1145/3077286.3077288.
- [25] S. Joshi, "Machine Learning Approach for Malware Detection Using Random Forest Classifier on Process List Data Structure." In: Proc. 2018. ACM Proceedings of the 2nd International Conference on Information System and Data Mining, pp. 98-102, 2018.
- [26] A. Bhattacharya, R. Tamal, and G. Kuntal, "A feature selection technique based on rough set and improvised PSO algorithm (PSORS-FS) for permission based detection of Android malwares," Int. J. Mach. Learn. Cybern., vol. 10, no. 7, pp. 1893–1907, 2019, doi: 10.1007/s13042-018-0838-1.
- [27] J. Park, H. Chun and S. Jung, "API and permission-based classification system for Android malware analysis," 2018 International Conference on Information Networking (ICOIN), 2018, pp. 930-935, doi: 10.1109/ICOIN.2018.8343260.
- [28] C. Gabriele, H. Aria, "Android Malware Detection Using Network Behavior Analysis and Machine Learning Classifiers," pp. 25-3, 2017, doi: 10.1007/978-3-030-90016-8_2.
- [29] A. R. Bhargav, J. C. Day, D. Steiner, "System and method for automated machine-learning, zero-day malware detection," U.S. Patent 9, pp. 292, 688, 2016.
- [30] H. Lee, D. Kim, M. Park, and S. Cho, "Protecting data on android platform against privilege escalation attack." International Journal of Computer Mathematics, vol. 93, no. 2, pp. 401-414, 2016, doi: 10.1080/00207160.2014.986113.
- [31] E. Cozzi, M. Graziano, Y. Fratanio and D. Balzarotti, "Understanding Linux Malware," 2018 IEEE Symposium on Security and Privacy (SP), 2018, pp. 161-175, doi: 10.1109/SP.2018.00054.
- [32] N. Ezzati-jivan and M. R. Dagenais, "Multi-scale navigation of large trace data: A survey,". Concurrency and Computation: Practice and Experience, vol. 29, no.10, pp. 1-20, 2018, doi: 10.1002/cpe.4068.
- [33] G. Meng, Y. Xue, Z. Xu, Y. Liu, J. Zhang, and A. Narayanan, "Semantic modelling of android malware for effective malware comprehension, detection, and classification," ISSTA 2016 - Proc. 25th Int. Symp. Softw. Test. Anal., pp. 306–317, 2016, doi: 10.1145/2931037.2931043.

- [34] A. Mahindru and P. Singh, "Dynamic permissions based android malware detection using machine learning techniques," *ACM Int. Conf. Proceeding Ser.*, pp. 202–210, 2017, doi: 10.1145/3021460.3021485.
- [35] M. Irshad, H. M. Al-Khateeb, A. Mansour, M. Ashawa, and M. Hamisu, "Effective methods to detect metamorphic malware: A systematic review," *Int. J. Electron. Secur. Digit. Forensics*, vol. 10, no. 2, pp. 138–154, 2018, doi: 10.1504/IJESDF.2018.090948.
- [36] M. Petraityte, A. Dehghantanha, and G. Epiphaniou, "Chapter 6 - Mobile Phone Forensics: An Investigative Framework Based on User Impulsivity and Secure Collaboration Errors," K.-K. R. Choo and A. B. T.-C. D. F. I. of C. and M. A. Dehghantanha, Eds. Syngress, 2017, pp. 79–89.
- [37] M. Salehi and M. Amini, "Android Malware Detection using Markov Chain Model of Application Behaviors in Requesting System Services," *IEEE Transactions on information forensics and security*, arXiv, pp. 1–11, 2017. doi: <https://arxiv.org/pdf/1711.05731.pdf>.
- [38] R.R. Ravula, K. J. Liszka, C.C. Chan, "Learning Attack Features from Static and Dynamic Analysis of Malware". In: Fred A., Dietz J.L.G., Liu K., Filipe J. (eds) *Knowledge Discovery, Knowledge Engineering and Knowledge Management. IC3K 2011. Communications in Computer and Information Science*, vol 348. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-37186-8_7.
- [39] A. Damodaran, F. D. Troia, C. A. Visaggio, et al. "A comparison of static, dynamic, and hybrid analysis for malware detection". *J. Comput. Virol. Hack. Tech.* vol. 13, pp. 1–12, 2017. <https://doi.org/10.1007/s11416-015-0261-z>
- [40] M. Leeds and T. Atkison, "Preliminary Results of Applying Machine Learning Algorithms to Android Malware Detection," 2016 International Conference on Computational Science and Computational Intelligence (CSCI), 2016, pp. 1070–1073, doi: 10.1109/CSCI.2016.0204.
- [41] N. McLaughlin et al., "Deep android malware detection," *CODASPY 2017 - Proc. 7th ACM Conf. Data Appl. Secur. Priv.*, pp. 301–308, 2017, doi: 10.1145/3029806.3029823.
- [42] T. Jia, L. Yang, P. Chen, Y. Li, F. Meng and J. Xu, "LogSed: Anomaly Diagnosis through Mining Time-Weighted Control Flow Graph in Logs," 2017 IEEE 10th International Conference on Cloud Computing (CLOUD), 2017, pp. 447–455, doi: 10.1109/CLOUD.2017.64.
- [43] J. Malik and R. Kaushal, "CREDROID : Android malware detection by network traffic analysis," *PAMCO '16: Proceedings of the 1st ACM Workshop on Privacy-Aware Mobile Computing*, July, 2016, pp. 28–36, doi: <https://doi.org/10.1145/2940343.2940348>.
- [44] H. Alimardani and M. Nazeh, "Permission-based Analysis of Android Applications Using Categorization and Deep Learning Scheme," *MATEC Web of Conferences*, 2019. doi: <https://doi.org/10.1051/mateconf/201925505005>.
- [45] H. Zhu, Z. You, Z. Zhu, W. Shi, X. Chen, and L. Cheng, "Neurocomputing DroidDet : Effective and robust detection of android malware using static analysis along with rotation forest model," *Neurocomputing*, vol. 272, pp. 638–646, 2018, doi: 10.1016/j.neucom.2017.07.030.

CHAPTER 4: MALWARE LAB EXPERIMENTAL DESIGN

4.0 Introduction

This chapter provides a brief description of the general lab setup and machine learning methods applications for cybersecurity. The chapter also discussed various tools used for the lab setup. Setting up a secure environment when working with a malicious data set is essential to avoid malware escalation to the outside environment. This chapter provides a general overview of how the lab was designed and its network was set up. The setup and configuration of the environment using virtual machines to avoid malware escalation to the outside environment was deployed using the principle of sandboxing. The sandboxing approach was used for safe analysis and execution of malicious files while preventing the host system and network from harm or infection. The rationale for using sandboxing approach was to add security layers to protect the host system and the network against malware escalation and other forms of attacks. This enabled the isolation of our lab from the external environment and safe execution of malicious files while observing and rating their activities and speeding up the malware identification process.

The deployment setup was configured based on the security principles and guidelines of preventing malware outbreaks to the outside environment. Some of the common practices and guidelines include: limiting the use of external devices such as USBs external hard drives on the host machines, user applications such as instant messaging which are capable of transferring malicious files were restricted, proscribed exchange of .exe and other related files through the dedicated email, prohibited the use of personal devices on the laboratory and the network of the University at large, and the use of content filtering and traffic monitoring tools as containment strategies.

4.1 Malware lab setup and tools

Effective examination of malware requires system contamination with malware samples or files and appropriate use of tools to monitor how such applications behave. As such, a securely configured laboratory is needed for the installation of malware tools for detection and analysis. When a proper and secured lab is designed either by virtualisation or by hardware approach, operational analysis can be performed without the production system being put at risk or affected.

This research adopted a virtualisation methodology for a flexible malware laboratory setup. The choice for virtualization methodology is its ability where the entire laboratory snapshots can be instantaneously taken. By this, the malware laboratory states can be easily recorded before and after infection. In addition, virtualisation methodology encourages easy editing of virtual machine settings properties such as memory, processors, Hard Disk, Network Adaptor, and even USB controller at any time of the experiment without affecting the state of any Virtual machine negatively. Finally, many virtual machines will be installed on a single physical machine as much as the investigator wants. The architectural build-up of the laboratory is standalone using VMware. VMware Workstation is a virtual environment for the installation and setting of VMs for malware analysis. In this thesis, each of the virtual machines was configured in a clean state of VMware workstation version 12.5.9 running on a physical machine (Windows 10). Virtual machines were configured during the lab setup. Two virtual machines are used in this study, namely, Windows 7 professional and ova REMnux version 6.0 (which is an Ubuntu distribution).

The choice for REMnux is because of its embedded tools for malware analysis and reverse engineering. Decoding, unpacking, disassembling, finding anomalies, and extracting malicious files are made easy when investigating malware with this tool. The ability of some of the tools to extract malware strings makes REMnux an interesting choice. The choice for these tools is because they are free, lightweight, and easy to configure on the production system which forms the selection benchmark. Other tools used for the lab include APKInspector, Androguard, Dex2jar, TcpDump, and virusTotal, among others. The choice for the selected tools is their compatibility with System Virtual Machines (SVMs). We choose SVMs because of the presence of a hypervisor that enables the complete execution of APK files, unlike the Process Virtual Machines (PVMs) approach.

4.1.1 APKInspector

APKInspector [1-4] is an open tool to aid an application's static analysis and to verify the behaviour of code in a visible mode. The tool inspects all the features in an application to access an Android device.

4.1.2 Androguard

Androguard [5-7] is a static analysis tool that provides an interactive interface when examining Android applications. This was used during APK DE compilation and disassembly to have access to the Android malware binary.

4.1.3 Dex2jar

Dex2jar tool [8-10] was used for the conversion of all the dex Android files into a readable java class file format for identification of the interrelationships with their components and representation creation times. By this, a detailed installation and activity timeline of the application can be examined.

4.1.4 Tcpdump

The inspection of both outgoing and incoming packets [11] associated with any mobile might become vitally significant especially those originating at the TCP layer of the network. Tcpdump [12-13] captures data about some useful information on malicious activities.

4.1.5 VirusTotal

This is a malware detection online tool that scans and analyses URLs and files. VirusTotal [14-6] contains many detection scanners such as AVG, Avast, Kaspersky, Avira, Qihoo, McAfee, Bitdefender, and TrustGo for analysis of suspicious files. In this research, the obtained dataset was scanned using this tool to determine its malicious and benign state. Other tools and libraries in the lab include WEKA, Python, and MATLAB.

4.2 Network setup

The experimental setting for this research was performed in a virtualised environment. We created and configured the virtual environment for experimentation by first setting up the network. Malware category such as worm has the functionality of being distributed to the external network and the production machine when the system is poorly configured. Worthy to note is that physical machine supports Network Address Translation for network traffic flow for virtual machines [17]; which when not controlled could be dangerous because malware could sneak through this network

connection to the outside world. Thus, causing unwanted operations to either the physical device, the network, or both. To avoid malware distribution, the network adaptor of the custom virtual machine was unbridged from connecting directly from the physical network but was connected as a custom to the virtual network by generating the IP address via the DHCP.

To achieve the aim and objectives of this experiment, all the virtual machines used (those running detection, analysis, and providing services) were connected through the LAN. However, all the VMs were detached from the host system and internet and were all connected to the Virtual Machine network. This gives malware a sense of network connectivity during live analysis but there is nothing resourcefully significant to which the network is connected to. The network configuration for the experimental design can be compared with the generic VM structure when traditional applications on the host device and the virtual machines on the guest OS are concurrently running. The host operating system runs the default traditional applications of the physical machine.

Custom configuration of the VMware network was implemented as the approach for VM configuration where all the VMs were set up to connect and use the same VMNet of the virtual switch. While the physical machine was connected to the internet (external network access), it was not connected to the VM executing malware. In order to enhance better network and power management of the VMs, the running systems were joined as a team of working VMs, and the network of the workstation was set on the host-only adaptor (see Figure 4-1).

The integration of the guest operating systems and the traditional application in the lab is illustrated in Figure 4-2. Each of the systems was installed and configured in a clean state, which means malware was not running on any of them during the configuration process. In summary, the system setup was configured in a clean VM state, and screenshots were taken at each state a data is captured and recorded. The network resolution of the workstation IP address was mapped to connect the virtual machines on the Host-Only Adaptor network terminal for network sharing from the external connection from the physical computer.

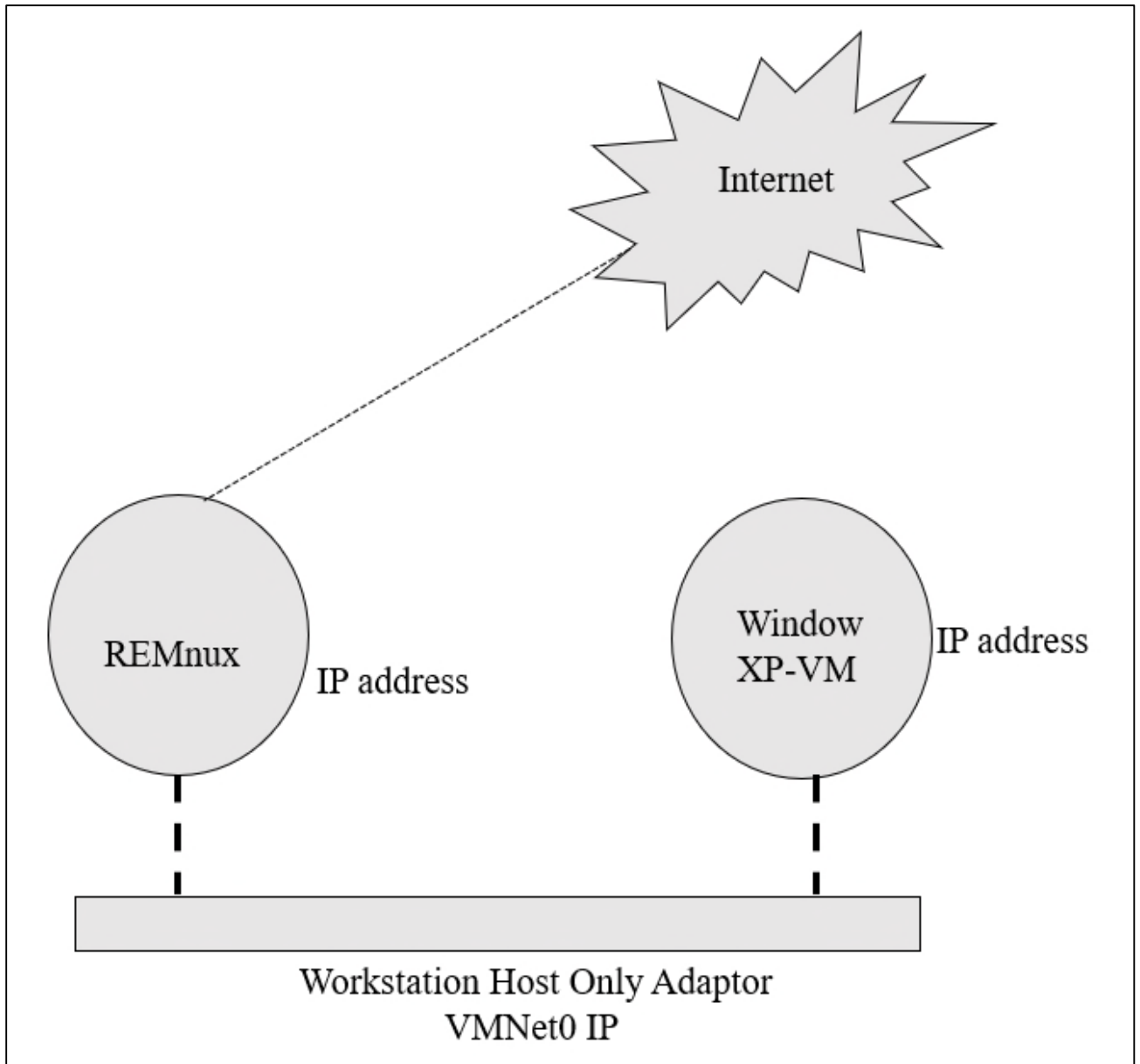


Figure 4-1 Setting VMs Ip address to host-only-adaptor

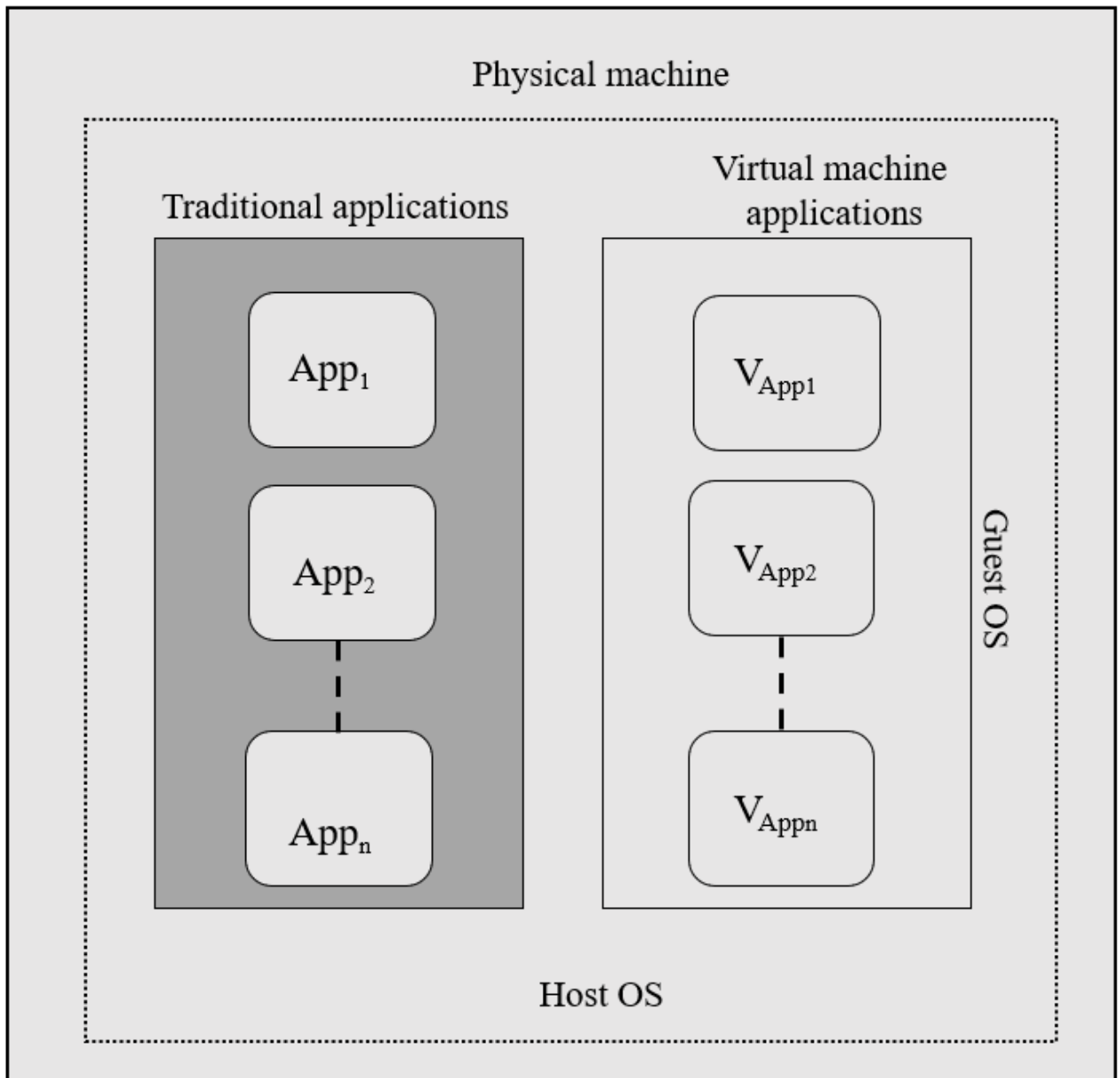


Figure 4-2 VM structure

4.2.1 The design components of the malware lab

The malware lab for this thesis is composed of different components running different functionalities and services. One of the components is the victim component that was pre-built from a clean disc image state before other services were added. The aim of setting up the victim is to examine the behaviour of the malicious sample while the malicious code is running. Another

component is a Linux-based server running other services to support the victim system during the behavioural and automated analysis of malware. Summary of the supported services includes but is not limited to the DNS and DHCP. The DNS redirects attempt made by malware for internet access, DHCP for getting IP addresses to redirect and trap malware attempts. Traffic capturing which is redirected by malware were enhanced by services such as Wireshark, FTP, HTTP, SMTP among others. Figure 4-3 shows the integrated setup of the physical and the virtual network of our lab.

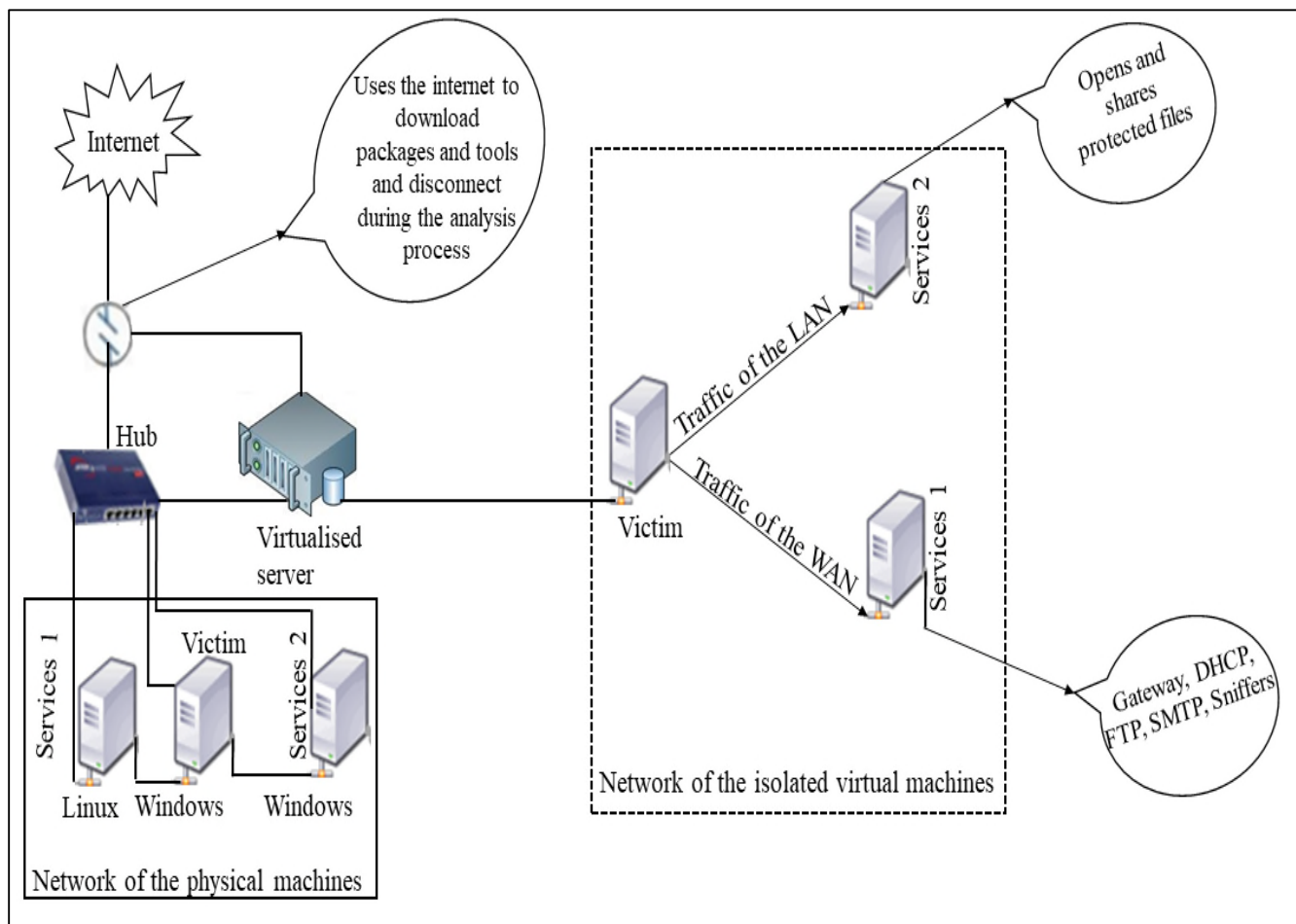


Figure 4-3 Integrated multi-Personal Computer in our malware lab

During the setup, we recognised the fact that some malicious codes do not run or may execute differently in a virtual environment. A hypervisor management system (Figure 4-4) was used for our lab setup to serve as the virtualisation pillar for all the experiments. This enhanced resource sharing to the virtual machines and enabled VMs interactions.

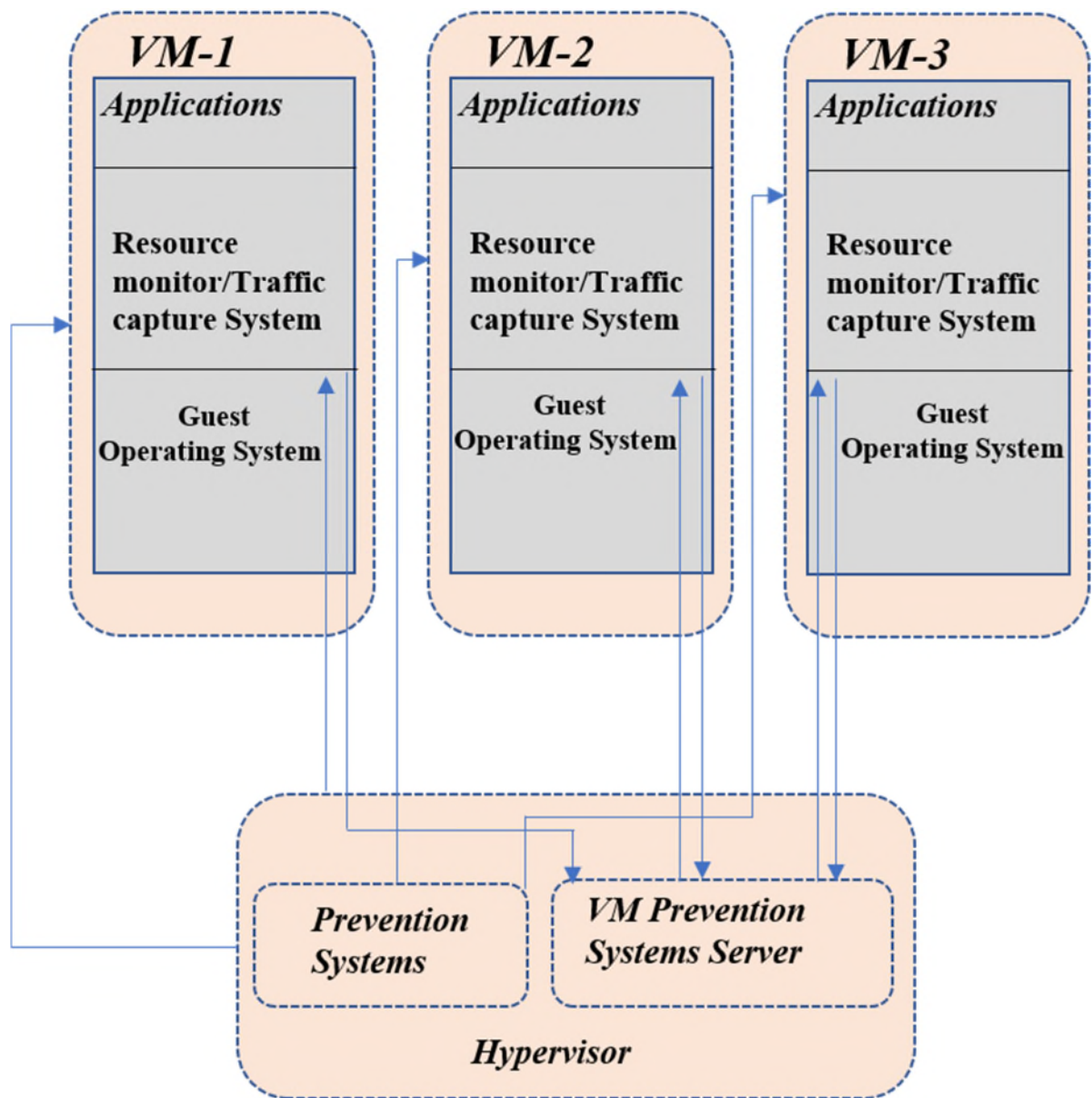


Figure 4-4 Virtualisation environment for the VM architecture of the hypervisor controlling component

4.3 Dataset deployment environment

Depositing or analysing malicious files in an unshielded environment is precarious to physical devices and network resources; especially when such environment is connected to production systems. When a secure environment is not primed before the malicious dataset is released,

malware escalation becomes inevitable. Malware escalation to the outside world can cause varying degrees of security contraventions [18] leading to violations of one or all the three most essential security components: confidentiality, integrity, and availability according to the research of [19-20]. Before releasing the dataset into the lab, both the automatic and the manual malware entry points were considered. Automatic entry points were done by using a dedicated email to send on the dataset from the repositories for analysis. Alternatively, an automatic entry point can be done through a honeypot. However, the researcher did not consider this approach since it requires a lot of time to set up the honeypot.

To avoid such occurrences, it was pertinent for our research to put adequate security measures and procedures in place while setting up the malware deployment environment. It is often recommended that such deployment be done in a virtualized environment with much premium to the safety of the surrounding resources to avoid accidental malware outbreaks. This setup was configured based on the security principles of the malware laboratory setup provided by [21-22] [23]. Other principles followed were those stated in [24] and [25] respectively. By following the laid down principles, will help bring malware escalation to the barest minimum. In this setup, two virtual machines were created. On each of the VMs, a 60 GB disk size was created for the installation and running of system resources. The virtual disk was installed as a single file for improving the system's performance. Though splitting the disk makes it easier to move the VM to other computers but then reduces the system performance and efficiency. The architecture of the deployment environment is illustrated in Figure 4-5.

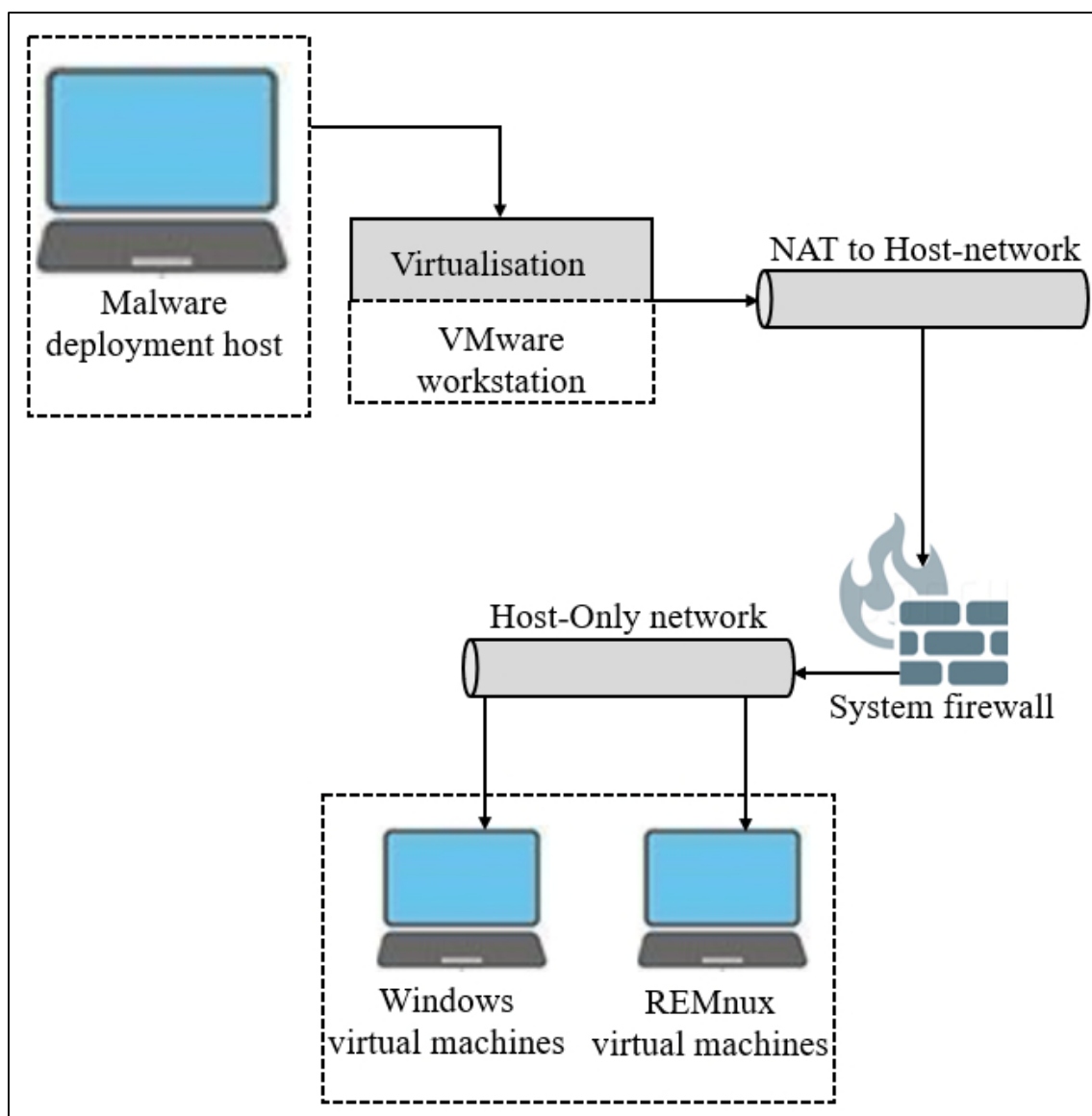


Figure 4-5 Malware deployment architecture of the lab design

The deployment host shown in the figure above is the physical system hosting all the virtual machines configured on the VMware Workstation. The host machine network connectivity was set to NAT while the networks of the VMs were set to Host-Only Network (HON) with a firewall between them. The firewall was configured to help monitor and filter both the inbound and outbound traffic. Traffic analysis of network logs using tools such as Wireshark and machine learning techniques will enhance the discovery of abnormalities of different malicious activities. VMware is used for machine learning virtualisation applications in different areas of cybersecurity by many organisations to reduce the cost of procuring thousands of servers.

4.4 Machine learning methods applications in cybersecurity

Depositing or analysing malicious files Machine learning (ML) incorporates a broad range of algorithms and modelling tools which are used for a vast array of tasks involving data pre-processing and other scientific tasks. Machine learning algorithms include but are not limited to Logistic Regression, Bayesian Additive Regression Trees, Support Vector Machines, Random Forest, among others. ML has impacted many areas of human endeavours such as industry including autonomous driving, manufacturing, the healthcare sector, energy, among others.

The overall ML goal is to recognise data patterns, which tell the manner hidden problems are handled. For example, in a complex system such as autonomous cars, the huge amount of data that comes from the sensor have to be converted into decisions on how such cars be controlled by a system that trained and understood the danger pattern. Machine learning techniques have been applied in several other areas of science including applications in cosmology, quantum computing, material physics, particle physics among others as discussed in the research of Carleo et al. [26].

Machine learning has become a vital technology for cybersecurity as well because of its exceptional features such as scalability, adaptability, and ability to tweak rapidly in both unknown and new challenges [27]. Different machine methods pre-emptively stamp out cyber threats and strengthen distinct security infrastructures. This is applied to pattern recognition, recognising anomalies detection, predictions, real-time cybercrime mapping, malware detection and analysis, among others. Just like other areas such as physical sciences, cybersecurity is one of the areas that are growing very rapidly because of the remarkable developments in social networks, online banking, mobile environments, web technologies, smart grids, among others. Different machine learning methods have been deployed to solve a variety of issues in cybersecurity such as phishing detection, intrusion detection, authentication, testing security implementation protocols, machine detection, among others. This subsection briefly describes some areas of the application of machine learning methods in different areas of cybersecurity.

4.4.1 Phishing detection

During phishing, the attacker aims at stealing or gathering sensitive data from the victim using deceptive approaches such as fraudulent emails and scammed websites. Machine learning techniques are applied as an anti-phishing detective, preventive, and corrective strategies. ML techniques are used in monitoring and filtering email content to identify and in most cases move the spam email to the spam box automatically. As a preventive and corrective approach to phishing, ML algorithms are developed for web application security, patch management, authentication, and forensics investigation accordingly.

Among others, the research of [28] designed fraudulent email detection using binary search feature selection. The proposed technique used the email subject, body, hyperlinks, and contents readability as the major feature sets for phishing detection. Zhuang et al. [29] applied an ensemble clustering approach to detect phishing emails using feature selection algorithms. In their approach, a hierarchical clustering algorithm was adopted for measuring the similarity between points. The machine learning method was used to detect phishing websites and malware samples.

4.4.2 Intrusion detection

Machine learning methods applications can also be seen in the network intrusion detection systems which is one of the vital areas of cybersecurity. For example, ML algorithms are designed to identify malicious network activities which cause availability, confidentiality, and integrity violations of systems networks. The use of intrusion detection systems (IDS) is viewed as one of the effective solutions for network security. Intrusion detection systems offer protection against different kinds of external threats even though the detection rate may be low under new threats and attacks. Intrusion detection is very essential in cybersecurity infrastructure to differentiate between attack and normal network access. Many of these detection tools are based on machine learning methods because of their adaptability in detecting new trends of attacks.

Different machine learning methods are applied in intrusion detection techniques, including but not limited to fuzzy logic [30], artificial neural network (ANN) [31], and support vector machine (SVM), among others. The study of Lu et al. [32] proposed a technique for misuse and anomaly detection on the network using a genetic algorithm. The machine learning-based framework detected 3,952 Neptune attacks and 4,000 intrusion connections.

Tao et al. [33] developed an IDS system based on a Support Vector Machine to detect network intrusions using crossover and mutation probability. The result of the simulation demonstrated that SVM and GA machine learning techniques accelerate intrusion detection with a lower positive rate. In [34], Chen et al applied the coarse-grained parallel genetic algorithm to optimise the parameters and feature subsets of SVM. In [35], SVM based machine learning methods were applied to develop a host-based intrusion detection system with the main security function to detect networking attacks by blocking and redirecting intruders using raspberry and SIEM servers as the main infrastructure requirements.

4.4.3 Breaking CAPTCHAs

Human interaction proofs usually called CAPTCHAs can be cracked by utilizing ML methods as demonstrated by Alqahtani and Alsulaiman [36]. The researchers deployed deep learning and machine learning algorithms including Naïve Bayes, random forest, classification and regression trees (CART), bagging with CART to break text and audio CAPTCHA. This shows the inadequacy of the human interaction proofs (HIP) at protecting services and systems from unauthorised users or hackers. Similarly, the research of Klecka [37] demonstrated how CAPTCHA features can be recognised using supervised machine learning techniques. The bubble captcha images were processed as input to the machine learning methods for optical character recognition of captcha codes.

4.4.4 Spam detection on social networks

Although machine learning methods can be exploited by different attacks such as causative attacks, exploratory attacks, targeted attacks, indiscriminate attacks, among others, it is still effective in spam detection on social networks. Lee et al. [38] noted that social systems are exploited by hackers in different forms. Their study developed a decoy system based on a Support Vector Machine for protecting social networks such as Twitter, Myspace and Facebook against those attacks. The social honeypot proposed solution represents a genuine user profile with a corresponding bot. Both spam and legitimate profiles are gathered by the model and then feeds them into the SVM based classifier.

4.4.5 Malware detection and classification

Malware detection and classification is another area that machine learning techniques are applied to address malicious attacks. Malware is one of the most challenging security issues in recent times. The number of malicious attacks keeps increasing which is too much volume for humans to handle. Machine learning uses various algorithms to analyse malicious datasets statistically to make assumptions about the nature of the malware file. The ability of machine learning to sort through millions of files and identify potentially malicious files or their binary equivalents has made ML uncover different kinds of malicious threats and automatically suppress them before they can cause any havoc. As a result, many businesses and organizations rely on machine learning to drive their cybersecurity products and services for early malicious threat identification, malware samples inspection and forensic analytics as demonstrated in the study of Usman et al. [39]. In [40], K-NN, SVM, and random forest machine learning algorithms were applied to detect malware on the Android operating system based on restricted API calls and filtered intents.

In summary, machine learning methods are crucial technological tools that are used in several areas of life including cybersecurity. As discussed in this section, there exist some robust and scalable anti-phishing machine learning algorithms and network intrusion and detection systems. Most existing authentication systems, security of human interaction proofs, data profiling, malware detection and analysis, and surveillance camera robbery detection are some of the examples machine learning can be applied in cybersecurity to address various security challenges. Even though machine learning assists in keeping many systems safe, ML classifiers are in themselves susceptible to different forms of attacks such as causative attacks which alter the training process and exploratory attacks which exploit the existing vulnerabilities.

REFERENCES

- [1] F. Jaafar, G. Singh and P. Zavarisky, "An Analysis of Android Malware Behavior," 2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C), 2018, pp. 505-512, doi: 10.1109/QRS-C.2018.00091.
- [2] M. K. Alzaylaee, S. Y. Yerima and S. Sezer, "Dynalog: an automated dynamic analysis framework for characterizing android applications," 2016 International Conference On Cyber Security And Protection Of Digital Services (Cyber Security), 2016, pp. 1-8, doi: 10.1109/CyberSecPODS.2016.7502337.
- [3] M. K. Alzaylaee, S. Y. Yerima and S. Sezer, "Dynalog: an automated dynamic analysis framework for characterizing android applications," 2016 International Conference On Cyber Security And Protection Of Digital Services (Cyber Security), 2016, pp. 1-8, doi: 10.1109/CyberSecPODS.2016.7502337.
- [4] A. Armando, U. Genova, G. Costa, U. Genova, S. Ranise, and R. Traverso, "Mobile App Security Assessment with the MAVeriC Dynamic Analysis Module,". In Proceedings of the 7th ACM CCS International Workshop on Managing Insider Security Threats, pp. 41–49, 2015, doi: <https://doi.org/10.1145/2808783.2808786>.
- [5] F. Mercaldo, C. A. Visaggio, G. Canfora and A. Cimitile, "Mobile Malware Detection in the Real World," 2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C), 14-22 May, 2016, Austin TX, Texas, pp. 744-746.
- [6] A. Atzeni, F. Díaz, A. Marcelli, A. Sánchez, G. Squillero and A. Tonda, "Countering Android Malware: A Scalable Semi-Supervised Approach for Family-Signature Generation," in IEEE Access, vol. 6, pp. 59540-59556, 2018, doi: 10.1109/ACCESS.2018.2874502.
- [7] S. Dong et al., "Understanding android obfuscation techniques: A large-scale investigation in the wild," Lect. Notes Inst. Comput. Sci. Soc. Telecommun. Eng. LNICST, vol. 254, pp. 172–192, 2018.
- [8] Y. L. Arnatovich, L. Wang, N. M. Ngo and C. Soh, "A Comparison of Android Reverse Engineering Tools via Program Behaviors Validation Based on Intermediate Languages Transformation," in IEEE Access, vol. 6, pp. 12382-12394, 2018, doi: 10.1109/ACCESS.2018.2808340.
- [9] R. Harrison, "Investigating the Effectiveness of Obfuscation Against Android Application Reverse," Tech. Rep. (Royal Hollow. Univ. London), vol. 7, no. March, pp. 0–63, 2015.
- [10] W. Chao et al., "An Android Application Vulnerability Mining Method Based On Static and Dynamic Analysis," 2020 IEEE 5th Information Technology and Mechatronics Engineering Conference (ITOEC), 12-14 June, 2020, Chongqing, China pp. 599-603, doi: 10.1109/ITOEC49072.2020.9141575.
- [11] C. Gandhi, G. Suri, R. P. Golyan, P. Saxena, and B. K. Saxena, "Packet Sniffer – A Comparative Study," Int. J. Comput. Networks Commun. Secur., vol. 2, no. 5, pp. 179–187, 2014.
- [12] Rudol, "Implementasi Keamanan Jaringan Komputer Pada Virtual Private Network (Vpn) Menggunkan," Implementasi Keamanan Jar. Komput. Pada Virtual Priv. Netw. Menggunkan Ipsec, vol. 2, no. 1, pp. 65–68, 2017.
- [13] J. Therdphapiyanak and K. Piromsopa, "An analysis of suitable parameters for efficiently applying K-means clustering to large TCPdump data set using Hadoop framework," 2013 10th Int. Conf. Electr. Eng. Comput. Telecommun. Inf. Technol. ECTI-CON, 15-17 May, 2013, Krabi, Thailand, pp. 1–6, 2013. doi: 10.1109/ECTICon.2013.6559650.

- [14] R. Masri and M. Aldwairi, "Automated malicious advertisement detection using VirusTotal, URLVoid, and TrendMicro," 2017 8th International Conference on Information and Communication Systems (ICICS), 4-6 April 2017, Irbid, Jordan, pp. 336-341, doi: 10.1109/IACS.2017.7921994.
- [15] I. Almomani and A. Khayer, "Android Applications Scanning: The Guide," 2019 International Conference on Computer and Information Sciences (ICCIS), 3-4 April 2019, Sakaka, Saudi Arabia, pp. 1-5, doi: 10.1109/ICCISci.2019.8716380.
- [16] K. Saikawa and V. Klyuev, "Detection and Classification of Malicious Access using a Dionaea Honeypot," 2019 10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS), 18-21 Sept. 2019, Metz, France, pp. 844-848, doi: 10.1109/IDAACS.2019.8924340.
- [17] D. Bekerman, B. Shapira, L. Rokach and A. Bar, "Unknown malware detection using network traffic classification," 2015 IEEE Conference on Communications and Network Security (CNS), 28-31 Sept. 2015, Florence, Italy, pp. 134-142, doi: 10.1109/CNS.2015.7346821.
- [18] S. Cobb and A. Lee, "Malware is called malicious for a reason: The risks of weaponizing code," 2014 6th International Conference On Cyber Conflict (CyCon 2014), 3-6 June 2014, Tallinn, Estonia pp. 71-84, doi: 10.1109/CYCON.2014.6916396.
- [19] D. Tobin and M. Stone, "Data Integrity: Recovering from a destructive malware attack," Natl. Cybersecurity Cent. Excell. Natl. Inst. Stand. Technol. May 2016, doi: <http://dx.doi.org/10.25675/10217/172988>.
- [20] G. R. Santhanam, B. Holland, S. Kothari and J. Mathews, "Interactive visualization toolbox to detect sophisticated android malware," 2017 IEEE Symposium on Visualization for Cyber Security (VizSec), 2-2 Oct. 2017, Phoenix, AZ, USA, pp. 1-8, doi: 10.1109/VIZSEC.2017.8062197.
- [21] K. Scarfone, "NIST Special Publication 800-83 Guide to Malware Incident Prevention and Handling for Desktops and Laptops," pp. 7, 2013.
- [22] A. Mohanta, A. Saldanha, "Malware Analysis Lab Setup. In: Malware Analysis and Detection Engineering". Apress, Berkeley, CA. https://doi.org/10.1007/978-1-4842-6193-4_2.
- [23] M. Rhode, L. Tuson, P. Burnap and K. Jones, "LAB to SOC: Robust Features for Dynamic Malware Detection," 2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks – Industry Track, 24-27 June 2019, Portland OR, USA, pp. 13-16, doi: 10.1109/DSN-Industry.2019.00010.
- [24] M. Sikorski and A. Honig, Practical malware analysis: the hands-on guide to dissecting malicious software. no starch press. 2012, pp. 9-167.
- [25] G. B. Bitjoka, "Network Malware Laboratory-Based On Honeypots Technologies," Journal of Cybersecurity Research (JCR), vol. 3, no. 1, pp. 1–12, 2018.
- [26] G. Carleo et al., "Machine learning and the physical sciences," Rev. Mod. Phys., vol. 91, no. 4, p. 45002, 2019, doi: 10.1103/RevModPhys.91.045002.
- [27] J. A. Carvajal Soto, F. Tavakolizadeh, and D. Gyulai, "An online machine learning framework for early detection of product failures in an Industry 4.0 context," Int. J. Comput. Integr. Manuf., vol. 32, no. 4–5, pp. 452–465, 2019, doi: 10.1080/0951192X.2019.1571238.
- [28] G. Sonowal, "Phishing Email Detection Based on Binary Search Feature Selection," SN Comput. Sci., vol. 1, no. 4, pp. 1–14, 2020, doi: 10.1007/s42979-020-00194-z.
- [29] W. Zhuang, Y. Ye, Y. Chen, T. Li, A. M. Categorization, and P. W. Detection, "Ensemble Clustering for Internet Security Applications," vol. 42, no. 6, pp. 1784–1796, 2012.
- [30] B. Farzaneh, M. Koosha, E. Boochanpour and E. Alizadeh, "A New Method for Intrusion Detection on RPL Routing Protocol Using Fuzzy Logic," 2020 6th International Conference on Web Research (ICWR), 2020, pp. 245-250, doi: 10.1109/ICWR49608.2020.9122278.

- [31] M. Pawlicki and M. Choras, “Neurocomputing Intrusion detection approach based on optimised artificial neural network,” vol. 452, pp. 705–715, 2021, doi: 10.1016/j.neucom.2020.07.138.
- [32] N. Lu, S. Mabu, and T. Wang, “An Efficient Class Association Rule-Pruning Method for Unified Intrusion Detection System using Genetic Algorithm,” pp. 164–172, 2013, doi: 10.1002/tee.21836.
- [33] P. Tao, Z. H. E. Sun, and Z. Sun, “An Improved Intrusion Detection Algorithm Based on GA and SVM,” *IEEE Access*, vol. 6, pp. 13624–13631, 2018, doi: 10.1109/ACCESS.2018.2810198.
- [34] Z. Chen, T. Lin, N. Tang, and X. Xia, “A Parallel Genetic Algorithm Based Feature Selection and Parameter Optimization for Support Vector Machine,” vol. 2016, 2016.
- [35] P. Nespoli, D. Díaz-lópez, and F. Gómez, “Journal of Information Security and Applications Cyberprotection in IoT environments : A dynamic rule-based solution to defend smart devices,” *J. Inf. Secur. Appl.*, vol. 60, p. 102878, 2021, doi: 10.1016/j.jisa.2021.102878.
- [36] F. H. Alqahtani and F. A. Alsulaiman, “Computers & Security Is image-based CAPTCHA secure against attacks based on machine learning ? An experimental study,” *Comput. Secur.*, vol. 88, p. 101635, 2020, doi: 10.1016/j.cose.2019.101635.
- [37] J. Klecka, “Recognition of CAPTCHA Characters by Supervised Machine Learning Algorithms,” *IFAC-PapersOnLine*, vol. 51, no. 6, pp. 208–213, doi: 10.1016/j.ifacol.2018.07.155.
- [38] K. Lee, J. Caverlee, and S. Webb, “Uncovering Social Spammers : Social Honeypots + Machine Learning,” In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, pp. 435–442, 2010.
- [39] N. Usman et al., “Intelligent Dynamic Malware Detection using Machine Learning in IP Reputation for Forensics Data Analytics,” *Futur. Gener. Comput. Syst.*, vol. 118, pp. 124–141, 2021, doi: 10.1016/j.future.2021.01.004.
- [40] V. Syrris and D. Geneiatakis, “On machine learning effectiveness for malware detection in Android OS using static analysis data,” *J. Inf. Secur. Appl.*, vol. 59, p. 102794, 2021, doi: 10.1016/j.jisa.2021.102794.

CHAPTER 5: HOST-BASED DETECTION AND ANALYSIS OF ANDROID MALWARE: IMPLICATION FOR PRIVILEGE EXPLOITATION

The content of this chapter is published in the International Journal for Information Security Research (IJISR).

Citation: M. Ashawa and S. Morris, “Host-Based Detection and Analysis of Android Malware: Implication for Privilege Exploitation,” *Int. J. Inf. Secur. Res.*, vol. 9, no. 2, pp. 871–880, 2019, DOI: <https://doi.org/10.20533/ijisr.2042.4639.2019.0100>.

The chapter is a preliminary study that builds on a technique known as Linux malware detect to address risks in our malware laboratory hosting environment. The purpose of building this in our laboratory is to have a track of detecting malicious activities on all the virtual machines accounts. As described in Chapter 4, our host systems comprise Windows and Linux machines. The Linux machine needs to be checked for threats that the Linux host machine is facing. For our research, this forms an intrusion detection system during analysis to extract active malware and their signatures targeting the shared hosts' machines. This model needs designing because most of the existing shared host scanners detect OS-level malware such as rootkits and trojans. However, they miss out on detecting other variants at the virtual user account level. The set of rules configured using LMD supported identification of malware activities including privileges escalated. The technique identifies activities that are threats when installed on a device which could result in serious security issues if not detected. This chapter has a link to the previous chapter by employing some of the strengths identified in the existing techniques. The preliminary experiment helped to determine the feasibility of applying some of the rules while developing the other detection methods.

Abstract

The Rapid expansion of mobile Operating Systems has created a proportional development in Android malware infection targeting Android which is the most widely used mobile OS. Factors such as the Android open-source platform, and low cost influence the interest of malware writers targeting this mobile OS. Though there are a lot of anti-virus programs for malware detection designed with varying degrees of signatures for this purpose, many do not give an analysis of what the malware does. Some anti-virus engines give clearance during installations of repackaged malicious applications without detection. This paper collected 28 Android malware family samples with a total of 163 sample datasets. A general analysis of the entire sample dataset was created given credence to their family samples and years discovered. A general detection and classification of the Android malware corpus were performed using a K-means clustering algorithm. Detection rules were written with five major functions for automatic scanning, signature activation, quarantine, and reporting the scan results. The LMD was able to scan a file size of 2048Mb and report accurately whether the file is benign or malicious. The K-means clustering algorithm used was set to 5 iteration training phases and was able to classify accurately the malware corpus into benign and malicious files. The obtained result shows that some Android families exploit potential privileges on mobile devices. Information leakage from the victim's device without consent and payload deposits are some of the results obtained. The result calls proactive measures rather than proactive in tackling malware infection on Android-based mobile devices.

5.1 Introduction

The evolution of information technology and its swift development of speedy transformation of wired and wireless communication in mobile computing especially smartphones have brought a proportional increase in malware development and attacks. Mobile devices today have played significant roles in our daily activities beyond making calls, sending SMS, and MMS. Financial transactions such as mobile banking and online shopping are readily made easy using mobile devices. The dramatic growth in popularity and usage of the Android OS is a result of their high computational abilities, openness, and low cost of the OS [1]. In addition, Android has created a significant impact on the permeation of broadband. Android has become a vulnerable operating system that is heavily targeted by malware writers across the broad spectrum of society.

There are a lot of detection techniques proposed and developed by different security researchers and mobile companies to fight this menace. However, none of the detection techniques and anti-virus engines proves to be 100% accurate in malware detection. As powerful as those detection mechanisms are, many have significant limitations in their analytical domain. The design of Android security is in a way that the running applications are isolated in the sandbox; thus, prohibiting direct interaction from the system assets such as SMS, address book, MMS, GPS.

Before an application is given access to the device resources during installation by the user, a proper verification check is performed by the binder to ensure that the application in quote meets the prerequisite for accessing the system resources. Other system resources such as activity manager, package manager, network state, launcher permission, and service monitor are also guided by the binder both at the user and kernel space. When a device is been infected, transactions at the binder driver are been intercepted by malware. This paper proposed a host-based detection of android malware by creating Linux malware detect rules on the virtual machine. This approach provides analytical information about the activities of the malware on the android device, it can also be applied to other types of the mobile operating system. This paper covers both old and newest Android malware samples and family distributions with a span of over four years.

5.2 Literature review

This section provided brief related literature on some of the related works that were carried out on the Android malware in this research. The dimensionality reduction approach used in the study of [2] extracted malware features when APK files were monitored under random projection. Manifest files and Dalvik [3] executables were filtered using a logistic classifier. The activities performed by the malware was however not significantly identifiable. However, there was a pragmatic substantial reduction within the large group of malware hashes with an initial step of pre-processing. Detecting android malware by looking at the device application packages; permission and system calls were performed in the study of [4].

The research of [5-7] looked at how malware can be avoided by looking at the installation permission spectrum and analysis of the application package before installation before the data availability in the application enhances detection with systematized prevalence [8-11]. The study of [12] investigated how Android malware can be detected by close surveillance for overlapping their static attributes. The study of [13-14] applied visualization and disassembly to inspect

Android malware chromatic similitude. The result shows that some Android malicious packages exhibit similar attributes to other families and have a high propensity of evading detection in virtual environments especially.

5.3 Methodology

The detection methodology defined in this work uses a dynamic approach, meaning that the malware sample was obtained and executed. The selected sample files were run in an isolated virtual environment and their behaviour was subjected to observation and analysis. Due to the danger of malware, the methodology was separated into different approaches ranging from installation and configuration of tools and services. Details of the methodological approach are provided in the subsections below.

5.3.1 System setup

The experimental design of this system was based on some components and services as illustrated in Figure 5-1. The setup lab was designed with open-source tools which can extract Android Apps libraries and resources. The safety of the physical machine was ensured by isolating it from the real machine and physical resources. The system setup and configuration were performed on Bionic Beaver which is an ubuntu codename for version 18.04 of the Ubuntu Linux-based operating system. The structure of the experiment was based on different security open-source tools which enhance application feature extractions during execution. The configured tools have the ability for scanning, monitor, and detect significant malware dynamic features ranging from battery consumption, GRP locations, and others.

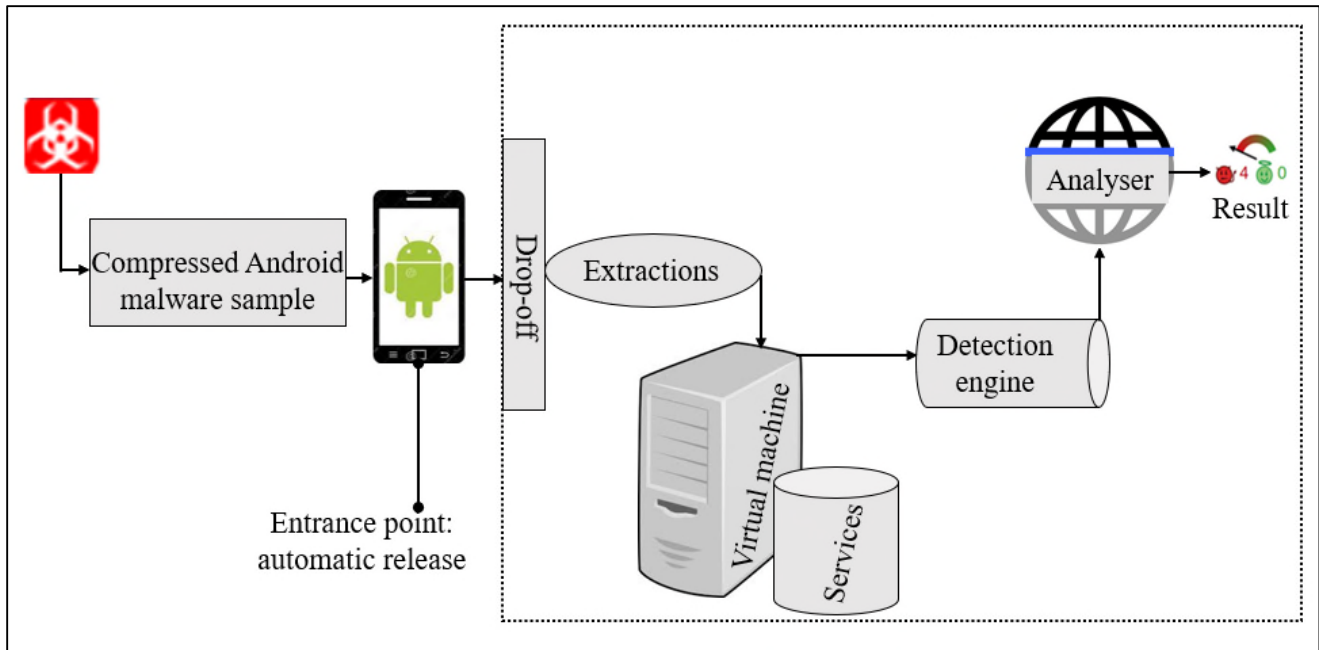


Figure 5-1 System setup for detection and analysis

5.3.2 Dependencies

For the configuration to achieve the research aim, some services were installed and configured on the virtual machine. The system was designed with dependencies for repackaging and unpacking Android applications running on the devices. Python version 3.6.5, ClamAV, volatility, chkrootkit, rhunter, and Linux malware detect (LMD) were installed as services for specific actions. The detected android malware file was then exported to the virus total for analysis. LMD rules scan malware and other threats in a host-based environment [15] and behavioural monitoring [16]. It uses both HEX and MD5 hashes to generate the required signature from ClamAV for malware features extraction which are derivatives of malware files from users. In this experiment, the rules were written in five different segments after download and installation. The written rules have the unlimited capability to detect and analyse 8,888 malware hashes of Trojans, viruses, rootkits, and other threats. The cron.daily dependency used in the configuration helps in updating daily emerging signatures with constant variable release and changelogs. The five most significant segments of the detection rules are summarily presented below (see Table 5-1 for details) which include: setting manual email scan reports for enabling malware alerts, signature files for scanning multiple weekly released malware with different signatures, scan options to specify maximum file

sizes that can be scanned per time interval, quarantine segment for suspending suspected files, and analysis mode for checking obfuscated files in an encoding mode.

Table 5-1 LMD rules

Functions	Rule Explanation
Setting manual email scan reports	# ' create an email address called christfavour783@gmail.com'. # [0 = disabled, 1 = enabled] email_alert="1" email_addr="christfavour783@gmail.com" email_subj="Android malware alerts for \$christfavour783 -\$(date +%Y-%M-%d)" email_ignore_clean="1"
Enable signature	# Enabled as new signatures a released multiple times per-week. autoupdate_signatures="1" autoupdate_version="1" autoupdate_version_hashed="1" cron_prune_days="21" import_config_url="" import_config_expire="43200" import_custsigs_md5_url="" import_custsigs_hex_url=""
Scan options	# The maximum directory depth that the scanner will search. scan_max_depth="15" scan_min_filesize="24" scan_max_filesize="2048k" scan_hexdepth="65536" scan_hexfifo="1" scan_hexfifo_depth="524288" scan_clamscan="1"scan_tmpdir_paths="/tmp /var/tmp /dev/shm /var/fcgi_ipc"
Quarantine	# The default quarantine action for malware hits quarantine_hits="1" quarantine_clean="1" quar_susp="1" quarantine_suspend_user="0" quarantine_suspend_user_minuid="500"
Analysis	#This is useful as obfuscated code is often stored using encoding methods string_length_scan="0" string_length="150000" clam av="1"

5.4 Machine learning algorithm

Machine learning [17] generally abbreviated as ML is a branch of Artificial Intelligence that deals with the computerization of data logical prototypical construction. ML enables systems to identify and classify data arrays, making decisions that are minimally based on the interference of humans. Machine learning can be supervised, unsupervised, reinforced, and semi-supervised learning. Supervised machine learning deals with training datasets to perform some regression or classification functions using algorithms such as decision trees, random forest Naïve Bayes, Nearest Neighbor, linear regression, and neural networks.

The unsupervised ML searches and divides the dataset of a particular sample based on its algorithm for mining, detecting, describing patterns for data grouping. The basic feature of this learning is that the target variables are usually not available. Commonly used unsupervised algorithms are associated rule and K-means clustering algorithms. This study used an unsupervised K-means algorithm to classify the malware corpus. K-means classification produced different clusters on the malware dataset at each iteration phase of training. During the experiments, the result of the data set classification tends to be more accurate as the number of iterations increases.

5.4.1 Experiment

A sizeable Android malware dataset was obtained from the Contagiodump project [18] for this study. The dataset was collected with the file name “Android-malware-master.zip” having a total size of 163 malware with 26 different malware families including benign (genuine) and malicious Android applications. Using Enguage Digitizer [19], the extracted components of the data set were graphically presented (see Figure 5-2) showing the dataset distribution of the malware families as generated. In the distribution, it is observed that there is a high degree of entropy in the sample size. Due to this visible distinctiveness in the elevation of these files’ high level of entropy, other applications will be considered for detection in the experiment, but much credence was given to the recent family to discover their determined characteristics.

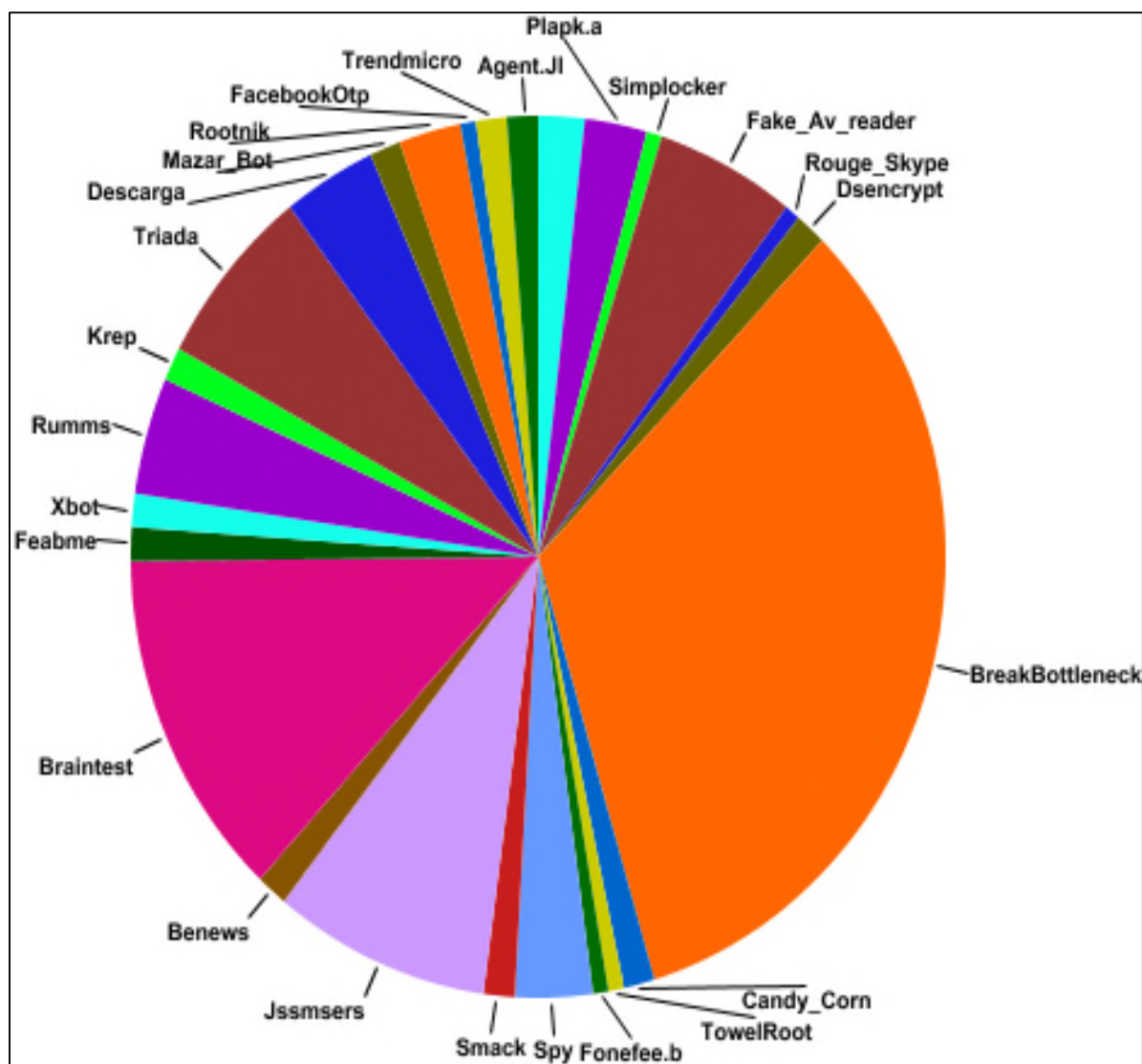


Figure 5-2 Distribution of the Android malware corpus using Engauge digitizer

5.4.2 Malware corpus timeline

It is very important to know when a malware family is emerged or discovered to determine when the payload effect on the vulnerable device has been patched. Using Androguard, the malware corpus timeline was carefully acquired by inspecting the 26 malware families within the 163 sample APK files. The APK files were carefully decompiled and extracted using APKTool as shown in Table 5-2.

Table 5-2 Dataset timeline consisting of Android malware families within the sample files

Android Malware	APKs	Discovered Date
Fave-Av-Reader	9	2013-08-31
Plapka	4	2013-09-07
Simplocker	1	2013-09-07
Rouge_Skype	1	2013-09-07
Dendroid	3	2013-12-20
Dsencrypt	2	2014-06-03
BreakBottleneck	54	2014-07-02
Candy_Corn	2	2014-01-26
TowelRoot	1	2014-11-30
Spy	5	2015-01-23
Simack	2	2015-02-21
Jssmsers	14	2015-02-21
Benews	2	2015-07-12
Braintest	21	2015-08-21
Feabme	2	2015-04-16
Xbot	2	2015-12-28
Rumms	7	2016-03-14
Krep	2	2016-05-23
Triada	11	2016-03-05
Descarga	6	2016-05-08
Mazar_Bot	2	2016-11-04
Rootnik	4	2016-11-24
FacebookOtp	1	2017-02-26
TrenMicro	2	2016-12-02
Agent.JI	2	2017-02-17
Fonefee.b	1	2017-04-20
Total samples	163	

5.4.3 Detection and feature extraction

Feature extractions from the dataset were done by installing the applications on the 6.0.1 Android version through the root directory of the detection engine. This required authentication to reload the system services upon the virtual machine for execution. The basic authentication required was the password (as shown in figure 5-3a), the permission that every application requires before installation on any device. After granting permission for application installations, the system detected immediately that a change in the system privacy modification was tempered with and generated a warning signal when the account check was performed (see figure 5-3b).

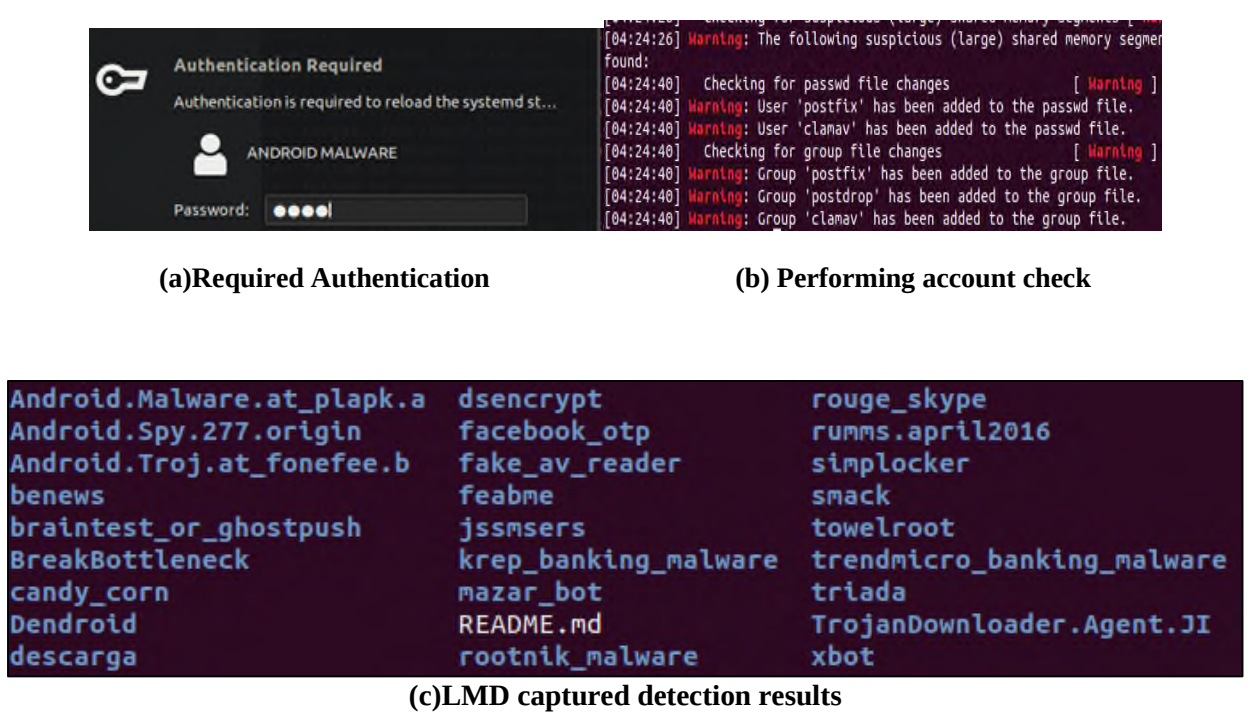


Figure 5-3 Authentication and detection

To find the similarity group in the Android malware data set and effectively classify the malware corpus, the K-means algorithm [20] was implemented in this work. This is an unsupervised machine learning algorithm used to group individual android applications based on similarity in the population. Our algorithm (K-means) works iteratively in this work to allocate data points based on the provided feature to the K group. Two conditions were specified in the algorithm. First, the centroids which are contained in K clusters should give a label to a new data, and secondly, the trained data label should be allocated to a single cluster. During the experiment, the K number of clusters was specified in the algorithm. Cluster 1 represented the group of benign

applications in the data set while cluster 2 represented the malicious applications for classification in the data set (see Figure 5-4).

Algorithm: K-Means clustering Algorithm

1. *Select K points, set of points $X_1 \dots X_n$*
2. *Randomly initialise the center point $C_1 \dots C_k$*
3. *Assign malware corpus to the closest cluster centroids*
4. *Compute the center point of each cluster*
5. **Repeat** steps 2 to 3 until no more change to the center points
6. *Exit*

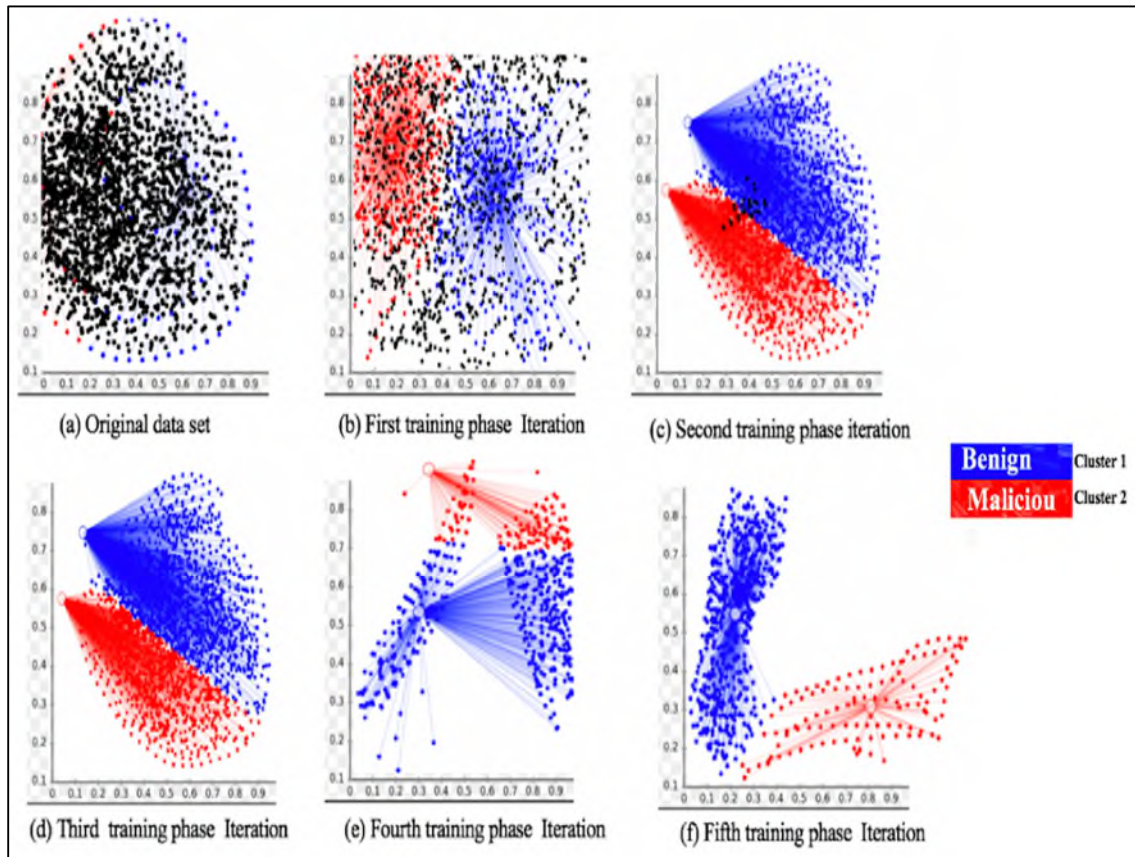


Figure 5-4 Classifying benign and malicious applications at each iteration

The algorithm was permitted to specify the closest, widest and marginal delta points at its space to observe and then append to each cluster at each iteration phase. The intention was to maximize the degree between the clusters and the given observation. If the delta point between a point to the two centroids is the same, it means the point could belong to the same group, thus classified as benign or malicious. At each iteration, the delta points between the clusters were systematically mirrored to group the dataset with similar attributes.

5.5 Metrics for performance evaluation

The following metrics were used for the performance evaluation of the classification system.

Accuracy: This shows the ratio of the overall integral value of the applications sample which is classified in a correct measure as malicious or benign.

Precision: This is the ratio of the actual malicious applications that are correctly classified or detected to the total number of the applications that the LMD model detected as malicious.

Recall: This is a ratio of true positive to its function and the additive value of the false-negative impact.

Confusion Matrix: The confusion matrix results obtained (see Table 5-3) summarised our classification model performance on the collected Android malware dataset. The true values with their respective derivatives are also represented accordingly. The integer of the malicious applications the LMD model correctly detected as malicious (True positive). The integer of the benign applications the LMD model incorrectly detected as malicious (false positive). Also, the integer of the benign applications the LMD model correctly classified as not malicious (True Negative). Finally, the integer of the malicious applications the LMD model incorrectly classified as not malicious (False Negative).

Table 5-3 Confusion matrix

Measure	Value	Derivation
Sensitivity	0.7027	$TPR = TP / (TP+FN)$
Specificity	0.7053	$SPC = TN / (FP+TN)$
Precision	0.73581	$PPV = TP / (TP+FP)$
Negative Predictive Value	0.6700	$NPV = TN / (TN+FN)$
False positive rate	0.29472	$FPR = FP / (FP+TN)$
False Discovery Rate	0.2642	$FDR = FP / (FP+TP)$
False Negative Rate	0.2973	$FNR = FN / (FN+TP)$
Accuracy	0.7039	$ACC = (TP+TN) / (P+N)$
F-Measure (F1 Score)	0.7189	$F1 = 2PT / (2PT+FP+FN)$

The relationship between the sensitivity and specificity (see Figure 5-5) in the random classification gives an illustration of the Android malware detection by a sequence in a custom analogous to the traditional Receiver Operating Characteristics (ROC) curve.

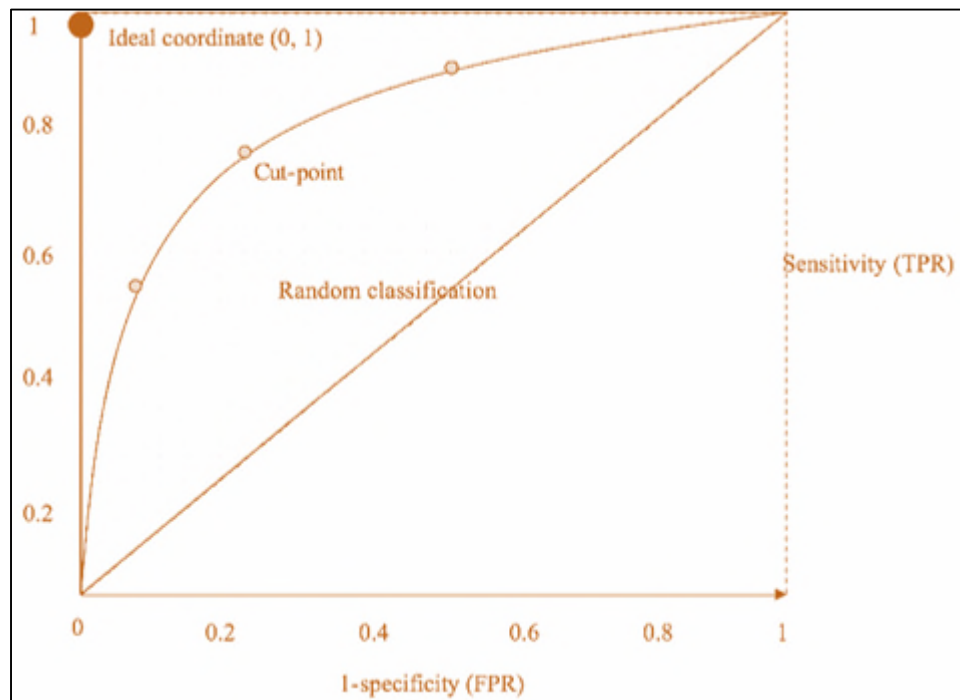


Figure 5-5 ROC curve showing classification accuracy

5.6 Evaluation

The result analysis for this research was focused on the latest discovered Android malware family (Fonfee.b) in the Dataset which has the package name **com.c101421042723.apk**. The choice to concentrate on Fonefee.b in this study is to give an in-depth study about the research. Most importantly, Fonefee.b is more recent and might not have been patched probably. Another factor of interest is the high level of randomness and the information gain at the leaf nodes of Fonefee.b file during the research. The analysis was focused on permissions requested, files accessed, contacted URLs, network connectivity, payload deposition, and battery depletion by the malware.

5.6.1 Permission request and URLs contacted

Malicious applications have little leverage on Android devices if their permission grants are controlled or constrained by users. The result obtained showed that the application requested different privileges when permitted to be installed in the root directory of the application. The research found that the malware established connectivity with different C&C servers where HTTP requests were made and contents such as images were uploaded at the background to those addresses without the victim's responsiveness (as shown in Figure 5-6a and 5-6b) respectively. Let us imagine that the infected mobile application is connected to implantable medical devices IMDs which hold valuable and sensitive information of patients or RFID devices. This could make an attacker initial communication to control patients' records or interfere with the devices during operation.

In this study, permissions requested by the malware and the URL contacted by the sample malicious application were shown in Figures 5-6a and 6b respectively. Intent filters by category indicated that the malware is a launcher with the ability to send SMS without the victim's knowledge. Other infection capabilities of this malware were discovered in the bundled files of the device' META-INF/MANIFEST.MF, popup, and automatic download are some special characteristics of this malware.

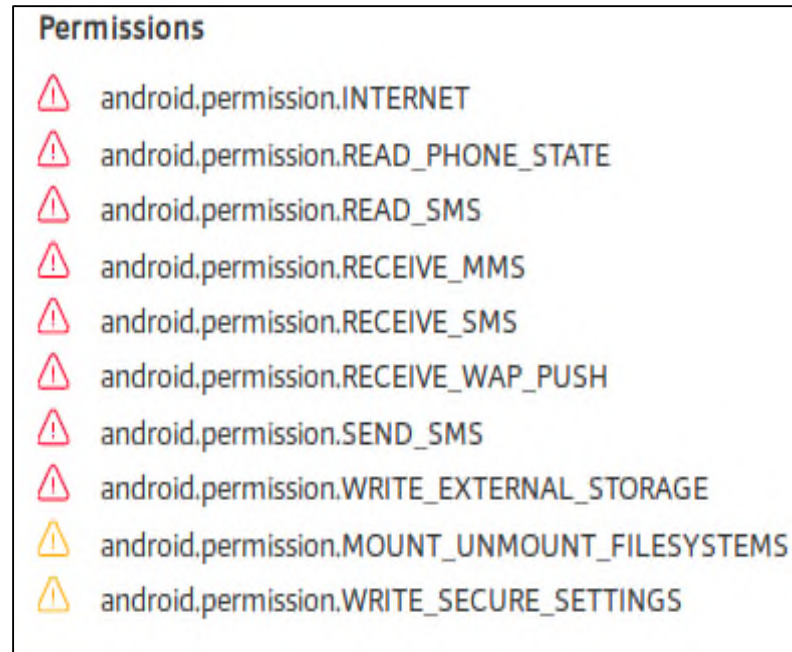


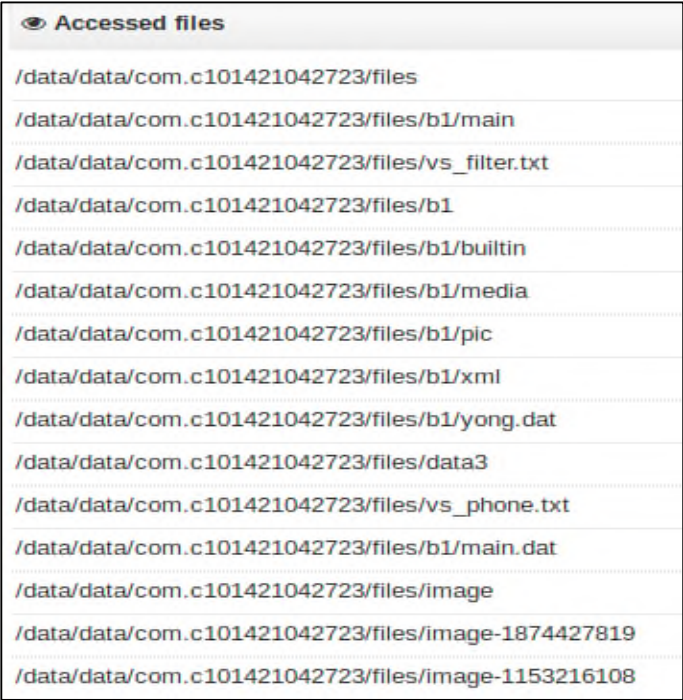
Figure 5-6a Permissions requested



Figure 5-6b URLs contacted, illustrating potential privileges exploited by Fonefee.b android malware

The permission requested remotely accessed files on the victim's device without his awareness. Accessed files include media, pictures, phone text messages, and images. The malware enabled a dynamic call method (android.net. ConnectivityManager.getMobileDataEnabled) which enables call APIs to provide access to information about the telephony services on the device. This has created a backdoor where other malicious applications can use such methods to determine the telephony services and states, as well as to access some types of subscriber information. From the

above figure (Figure 5-7), the commonly accessed files included but were not limited to picture files, text files, and media files.



Accessed files
/data/data/com.c101421042723/files
/data/data/com.c101421042723/files/b1/main
/data/data/com.c101421042723/files/vs_filter.txt
/data/data/com.c101421042723/files/b1
/data/data/com.c101421042723/files/b1/builtin
/data/data/com.c101421042723/files/b1/media
/data/data/com.c101421042723/files/b1/pic
/data/data/com.c101421042723/files/b1/xml
/data/data/com.c101421042723/files/b1/yong.dat
/data/data/com.c101421042723/files/data3
/data/data/com.c101421042723/files/vs_phone.txt
/data/data/com.c101421042723/files/b1/main.dat
/data/data/com.c101421042723/files/image
/data/data/com.c101421042723/files/image-1874427819
/data/data/com.c101421042723/files/image-1153216108

Figure 5-7 Files accessed by fonefee.b malware

5.6.2 Network communication

As shown in Figure 5-8, Fonefee.b was found to be secretly communicating and uploading pictures to the particular IP address of a country. The malware loaded and invoked method android.net.ConnectivityManager.getMobileDataEnabled for background data connectivity and android.net.conn.CONNECTIVITY_CHANGE for synchronization mechanisms respectively. In addition, background calls, uploading of images, and media files were made through the influence of the malware to a country IP address (name on hold for security reasons). The relationship between the contacted domains, compressed parents, bundled files, and the contacted IP address of the country was obtained.

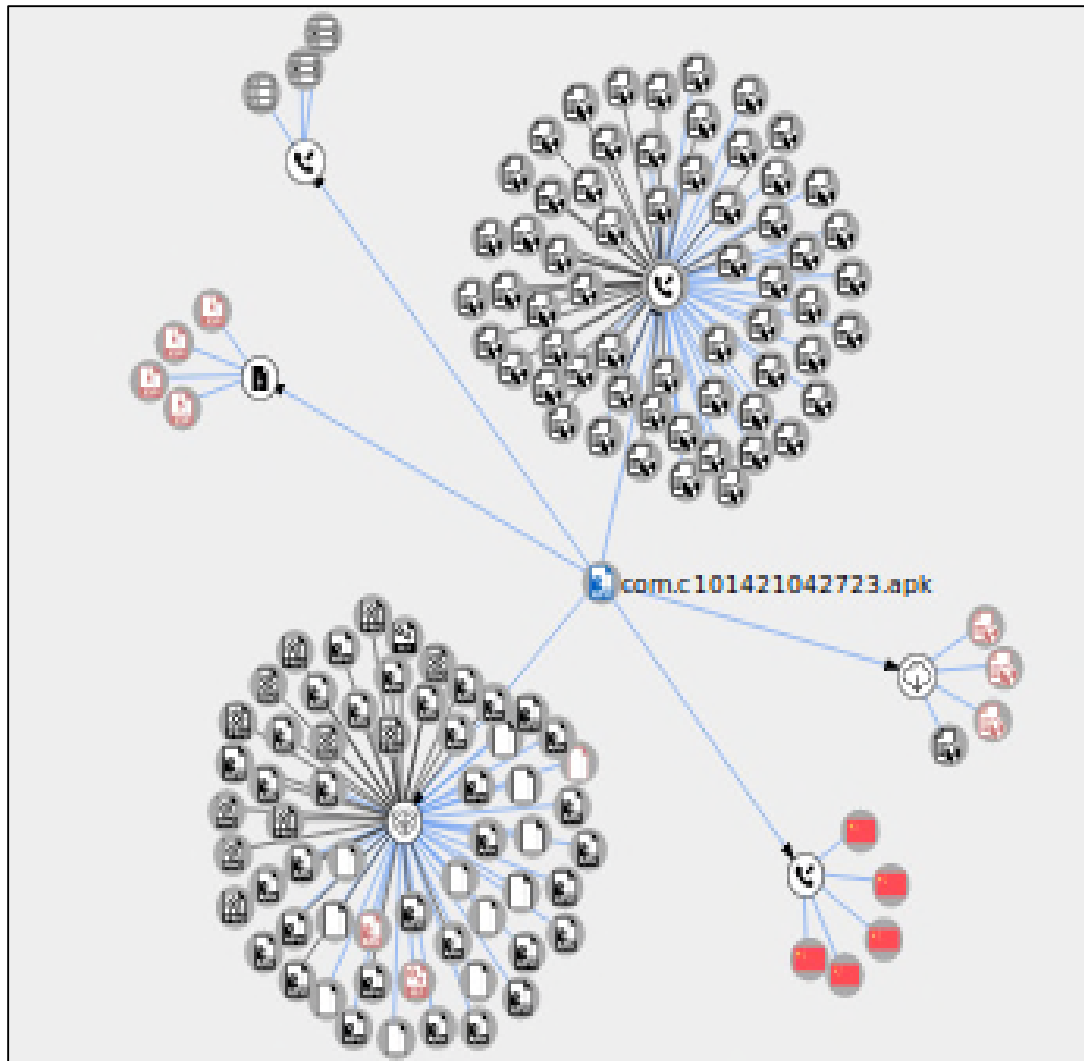


Figure 5-8 Malware activity connectivity

5.6.3 Payload and location monitoring

Figures 5-9 and 5-10 show the payload and location monitoring activities of the malware on the infected device respectively. Payload is another interesting feature to look for when analysing malware. These are private text containing malware for victims' data encrypted deletion and spamming. Fonefee.b deposited a payload in the device with an encrypted submit name filename"cd9404f21e4bc52e477e62037db537c4239b9afcf1d490e1ea75d1c5aefb47ef". This was primarily for performing phishing attacks while resident in the device. Part of the message was obtained during the analysis which was executed in the particular offset of the device address.

ELF sections					
Name	Type	Address	Offset	Size	Flags
	NULL	0x00000000	0x00000000	0	
.dynsym	DYNSYM	0x00000114	0x00000114	1424	A
.dynstr	STRTAB	0x000006a4	0x000006a4	1517	A
.hash	HASH	0x00000c94	0x00000c94	632	A
.rel.dyn	REL	0x00000f0c	0x00000f0c	328	A
.rel.plt	REL	0x00001054	0x00001054	264	A
.plt	PROGBITS	0x0000115c	0x0000115c	416	A, X
.text	PROGBITS	0x000012fc	0x000012fc	77788	A, X
.ARM.extab	PROGBITS	0x000142d8	0x000142d8	396	A
.ARM.exidx	ARM_EXIDX	0x00014464	0x00014464	2224	A, L

Figure 5-9 Fonefee.b Payload

From the obtained results, the activity of the Fonefee.b malware family was present on the host device. The family of the malware is a Trojan capable of monitoring and tracking the victim's device by taking the coordinates of the victim's latitude. At the time of this experiment, the location of the device was captured at latitude 51.5595N and 1.7910W respectively. This ability of the Trojan to track the device location and send the details to the C&C server of the hacker can be used to launch terrorism or personal attacks on the individual. This is indeed a dangerous Trojan exhibiting almost all the actions performed by the rest of the malware class.

```
T\BB\9C\F0\93\E5,\E3Na_C^\A0\B6Nd\89\BA\B5\F4\
\C4\8D\FB\DA\C5\F7E#Hr\AB\BB\FD\EB\C1S\8F^\
2\99\C7U\FC\CCC\A7\8B\90\D5d\A9\C1\E2
\BFE\A1l\A4\D2\E6\U\BB4\94\K\96$@\Andro
\Android.Troj.at_fonefee.b\p7)\E9\H4:\E8\BB\
d.Tro \B6\B2\9F\DB\ED'\F7<\F8p?6\B2\9DCmY3س
\95\banking_malware95D\B7\E2\D5\Q\86\8B\DCP\
-o\B1\D46\E4T\F1\A7\CDE0\878_/;gY,\BF\BDE'\D2c/
Ai\DA^\BC\E7=\EEL\94X\89*>w!u\B0\BFHr\00\8D=q
0\A8\A1w\D7~51.5595N''swindon-'1.7910W?15183\9
A6\D6\F4\9FL\9A\9D\F5\B4U5\B2\F8\AD\9DH\D7#F\
98\ACe\A5cY\B1\E4,\B8\9DcN\B4\DA'\CB\EC\BAB9
```

Figure 5-10 Location monitoring

5.6.4 Battery depletion

The consumption rate per activity was monitored using a volatility tool. The ratio of energy consumption to the bandwidth was not proportional due to the draining effect exerted by the malware. The battery depletion was a range of many activities performed in the background by the malware. The battery life of a mobile device is determined by the rating of the input current of the phone in relation to the load current the circuit holds. By inference, the battery capacity of a mobile device under normal conditions will be high when the load activities on the system are low. However, some external forces such as malware make allowances in which the device battery life can be affected especially when the runtime is not equivalent to the ratio of load per second running on the device.

5.7 Conclusion

The results obtained from this study confirmed visibly the sombre menace malware has posed on the Android platform operating system. There are many anti-virus programs modelled to tackle this threat. However, the empirical examination of our research revealed that many of these software lacks efficacies in detecting malware that emerged with unknown signatures by the existed anti-virus programs. Fonefee.b was only detected by 38 out of 59 anti-virus engines. The rest of the engines failed to extract the signature of the sample malware. In addition, more than one cryptographic function and hash values were used while writing this malware program. By implication, this can exhibit metamorphic attributes to evade detection by many detection techniques through self-repackaging.

Exploitation to gain privilege escalation as a result of the fragmentation problem observed in the Android operating system can enhance mobile vulnerability risk before a security patch is initiated. It is observed in this research that Android lacks firm rheostat on some of the APIs such as “sendTextMessage” to hinder premium rate sending of SMS at the background when affected with malware. Most of the permissions permitted by malicious applications can affect the entire Android OS configuration and META-INF/MANIFEST.MF. All the potentially exploited privileges were a function of this vulnerability. Some of the domains that were contacted include lm.dmsd.net, sdk2.cmvideo.cn, and vsbbc.com using a mobile communication address 221.181.100.84 of the country’s network.

In conclusion, the results obtained and presented in this paper provided detailed characteristics and threats that malware posed on users of Android-based devices and information security at large. These threats range from malware payload deposition, information harvest, privilege escalation, private communication without the victim's awareness, upload of the device's data contents to C&C servers, and a host of others. These outcomes are indeed graving and call for advanced security measures to be implemented to provide a reactive defence mechanism on the OS rather than proactive defence. Future research can be carried out using neural network algorithms such as Bayesian regularisation to determine the accuracy rate of both the target and output of the malware classification sample. The researcher intends to carry further research on how to classify permissions requests based on their threat and protection level. Further research can be carried out on how to provide a security perimeter defence around the Google Bouncer to fight Android malware infections.

Acknowledgement

This project was fully funded by the Petroleum Technology Development Fund (PTDF) of Nigeria (No. PTDF/ED/PHD/AMA/1245/17).

REFERENCES

- [1] H. Zhu, Z. You, Z. Zhu, W. Shi, X. Chen, and L. Cheng, "Neurocomputing DroidDet : Effective and robust detection of android malware using static analysis along with rotation forest model," *Neurocomputing*, vol. 272 no.6, pp. 638–646, 2018, doi: 10.1016/j.neucom.2017.07.030.
- [2] L. Sayfullina, E. Eirola, D. Komashinsky, P. Palumbo and J. Karhunen, "Android Malware Detection: Building Useful Representations," 2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA), 18-20 Dec. 2016, Anaheim, CA, USA, pp. 201-206, doi: 10.1109/ICMLA.2016.0041.
- [3] H. Jiang, H. Y. B, S. Qin, Z. Su, and J. Zhang, "Detecting Energy Bugs in Android Apps Using," In: Duan Z., Ong L. (eds) *Formal Methods and Software Engineering. ICFEM 2017. Lecture Notes in Computer Science*, vol. 10610. Springer, Cham, doi: https://doi.org/10.1007/978-3-319-68690-5_12.
- [4] E. Caron, L.L.C., McAfee, "Method and processes for securely autofilling data fields in a software application. U.S. Patent 9,766,957, 2017.
- [5] N. Chiluka, A. K. Singh and R. Eswarawaka, "Privacy and Security Issues Due to Permissions Glut in Android System," 2018 International Conference on Inventive Research in Computing Applications (ICIRCA), 11-12 July 2018, Coimbatore, India, pp. 406-411, doi: 10.1109/ICIRCA.2018.8597321.
- [6] C. Tansettanakorn, S. Thongprasit, S. Thamkongka and V. Visoottiviseth, "ABIS: A prototype of Android Botnet Identification System," 2016 Fifth ICT International Student Project Conference (ICT-ISPC), 27-28 May 2016, Nakhonpathom, Thailand, pp. 1-5, doi: 10.1109/ICT-ISPC.2016.7519221.
- [7] M. Yusof, M. M. Saudi, and F. Ridzuan, "A new mobile botnet classification based on permission and API calls," *Proc. - 2017 7th Int. Conf. Emerg. Secur. Technol. EST 2017*, pp. 122–127, 2017, doi: 10.1109/EST.2017.8090410.
- [8] Y. Zhou and X. Jiang, "Dissecting Android Malware: Characterization and Evolution," 2012 IEEE Symposium on Security and Privacy, 20-23 May, 2012, San Francisco, CA, USA, pp. 95-109, doi: 10.1109/SP.2012.16.
- [9] J. Chen, "Detecting Android Malware Using Clone Detection," *J. Comput. Sci. Technol.*, vol. 30, no. 5, pp. 942–956, 2015, doi: 10.1007/s11390-015-1573-7.
- [10] Z. Wang, C. Li, Y. Guan, Y. Xue and Y. Dong, "ActivityHijacker: Hijacking the Android Activity Component for Sensitive Data," 2016 25th International Conference on Computer Communication and Networks (ICCCN), 1-4 Aug. 2016, Waikoloa, HI, USA, pp. 1-9, doi: 10.1109/ICCCN.2016.7568487.
- [11] S. -W. Hsiao, Y. S. Sun and M. C. Chen, "Behavior grouping of Android malware family," 2016 IEEE International Conference on Communications (ICC), 22-27 May 2016, Kuala Lumpur, Malaysia, pp. 1-6, doi: 10.1109/ICC.2016.7511424.
- [12] M. Nezhadkamali, S. Soltani and S. A. H. Seno, "Android malware detection based on overlapping of static features," 2017 7th International Conference on Computer and Knowledge Engineering (ICCKE), 26-27 Oct. 2017, pp. 319-325, Mashhad, Iran, doi: 10.1109/ICCKE.2017.8167899.
- [13] W. Park, K. -H. Lee, K. -S. Cho and W. Ryu, "Analyzing and detecting method of Android malware via disassembling and visualization," 2014 International Conference on Information and Communication Technology Convergence (ICTC), 22-24 Oct. 2014, Busan, Korea (South), pp. 817-818, doi: 10.1109/ICTC.2014.6983300.
- [14] W. Markus, F. Fischer, R. Luh, A. Haberson, A. Rind, and A. Keim, "A survey of visualization systems for malware analysis," In *EG Conference on Visualization (EuroVis)-STARs*, 29 May, 2015, Cagliari, pp. 105-125, 2015, doi: <https://dx.doi.org/10.2312/eurovisstar.20151114>.

- [15] Linux Malware Detect - R-fx Networks, 2017. Accessed on: Nov. 27, 2018. [Online] Available: <https://www.rfxn.com/projects/linux-malware-detect/>.
- [16] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, and Y. Weiss, “‘Andromaly’: a behavioral malware detection framework for android devices,” *Journal of Intelligent Information Systems* vol. 38, no.7, pp. 161-190, 2012, doi: 10.1007/s10844-010-0148-x.
- [17] W. Andrew, P. K. Ferentinos, and P. G. Petropoulos, "A new synergistic approach for monitoring wetlands using Sentinels-1 and 2 data with object-based machine learning algorithms," *Journal of Environmental Modelling & Software*, vol. 6, no 104, pp. 40-54, 2018.
- [18] E. Yu. "Contagiodump project" Android Malware Samples. Accessed on: Dec. 18, 2018. [Online]. Available <https://github.com/ashishb/android-malware/find/master>.
- [19] M. Mitchell, B. Muftakhidinov, T. Winchen, “Engauge Digitizer”. Accessed on: Dec. 18, 2018. [Online]. Available: <http://markummitchell.github.io/engauge-digitizer/>.
- [20] Y. Chunyong, and S. Zhang, "Parallel implementing improved k-means applied for image retrieval and anomaly detection," *Journal of Multimedia Tools and Applications*, vol. 76, no. 16, pp. 16911-16927, 2017.

CHAPTER 6: ANDROID PERMISSION CLASSIFIER: A DEEP LEARNING ALGORITHMIC FRAMEWORK BASED ON PROTECTION AND THREAT LEVELS

The content of this chapter is published in the International Journal for Security and Privacy, Wiley Publisher.

Citation: M. Ashawa and S. Morris, “Android Permission Classifier : a deep learning algorithmic framework based on protection and threat levels,” Wiley Security and Privacy³, vol. 4 .no 5, 2021, pp. 1–26, 2021, DOI: 10.1002/spy2.164.

Android permission features are very significant in both static and dynamic analysis for malware detection and classification process. Permissions are often used by both malicious and benign applications to access the resources of a system. Dangerous permissions, when granted access during the application’s installation, can cause a lot of problems to the device and its resources. As a result, there is a need to build a classification technique to classify Android applications based on their permissions declared by both benign and malicious Android apps. Existing permission models used limited information and permissions for classification. Our method used information gained to classify dangerous permissions from over 100 permissions and other attributes using a large data set. Our model has a good feature diversity capacity which is significant in gathering all dangerous permissions fine-grained features to detect different malware variants. The framework is essential to enhance privacy and security protection for the mobile community. This chapter is linked to objective 4 and the research question. This chapter is an integration of the techniques' strengths identified in chapter 3 and their successful application in chapter 5. The techniques were leveraged to build an automated framework for analyzing smart and sophisticated malware based on their feature attributes. This chapter is significant because the framework provides a platform for large-scale automated analysis of smart and emerging malware attributes which are used as attack vectors to infect Android-based devices.

Abstract

Recent works demonstrated that Android is the fastest growing mobile OS with the highest number of users worldwide. Android's popularity is facilitated by factors such as ease of use, open-source, and cheap to purchase compared to mobile OS like iOS. The widespread of Android has brought an exponential increase in the complexity and number of malicious applications targeting Android. Malware deploys different attack vectors to exploit Android vulnerability and attack the OS. One way to thwart malware attacks on Android is the use of Android security patches, antivirus software, and layer security. However, the fact that the permission request dynamic is different from other attack vectors, makes it difficult to identify which permission request is malicious or not especially when constructing permission request profiles for Android users. The aforementioned challenge is tackled by our research. This article proposed a framework called Android Permission Classifier for the classification of Android malware permission requests based on threat levels. This article is the first to classify Android permission based on its protection and threat levels. With the framework, out of the 113 permissions extracted, 23 were classified as more dangerous. Our model shows classification accuracy of 97% and an FPR value of 3% with high diversity capacity when compared with the performance of other similar existing methods.

Keywords: Android permission request, feature diversity, machine learning algorithms, neural networks.

6.1 Introduction

Smartphones' increase in the market for business and personal purposes has brought about a corresponding increase in the number of malicious applications today. Malware attacks on mobile devices have become one of the most dreaded and precarious threats causing economic and information impairments. The study of Ashawa et al [1-2] asserted that the increase and fast-growing rate at which mobile applications are been used in the 21st century has caused a proportionate growth in the number of cyber-attacks one of which is malware attacks on mobile and personal computers. Though malicious applications target different mobile OS platforms, malware targeting Android has become a massive surge recently.

Smartphone market share recently published by the International Data Corporation (IDC) reported that Android OS shipment worldwide has slightly increased from 85.1% in 2018 to 86.7% in the second quarter of 2019 [3] and is the most popular mobile OS. The popularity of Android and its fast-growth shipment worldwide made the OS the major attack target by malware writers. This proliferation of different app from third parties makes application repacking and decompilation easier on the Android platform than could be done on the iOS platform due to Android's open-source. Android popularity has made the most targeted mobile OS by mobile malware according to the recent research published by Kim et al [4]. Some of the factors which enhanced the popularity of Android OS include but are not limited to open-source, cheap to purchase, easy to use, and the launching of new models due to 5G inventions. Among other factors, open-source makes it easier for a lot of android developers to publish their applications on different Android market platforms without much critical security analysis and scrutiny, unlike iOS.

Different detections, classifications, and analysis techniques have been widely proposed by various researchers and security organizations across the world to curb the widespread of malware and to alleviate their attacks on Android devices. Some of the proposed techniques and strategies include dynamic detection [5-12] static detection [13-17] hybrid detection [18-22], and memory forensics techniques [23-31] respectively. The technique observes and monitors the application's behaviour by focusing on knowing how the malware is connected and is interacting with mobile device resources at run time. The Static detection technique involves malware analysis without execution of the malware application but focuses on the evaluation of the signature and heuristic behaviour of the application using semantic properties.

The hybrid technique is an amalgamation of both the static and dynamic technique features deployed for malware detection. It moderates at runtime portions of malware code to be inspected using the static technique. Malware detection by the hybrid approach creates more stability in detection efficacy and accuracy by using advantages from both dynamic and static methodologies to provide more detection results and efficiency. The memory forensics analysis technique involves acquiring the device memory image which is infected or suspected using forensics memory acquisition tools and then analysing the memory dump for forensic artefacts such as running processes, list of network connections, kernel modules, list of shared libraries, and rootkits. The technique is more reliable and efficient [32] when investigating sophisticated malware such as polymorphic and metamorphic.

Several types of research had been carried out on android malware detection, classification, and analysis. Our research is different from the rest because there has not been any research that focused on extracting and classifying Android malware permission requests based on their protection and threat levels using deep learning. While there are four permission attributes namely, normal permissions, dangerous permissions signature permissions, and signatureOrSystem permissions, our research focuses on the first two which are the normal and dangerous permissions. Android applications have many features including strings, opcode frequencies, API features, shared libraries, and so on which many previous studies have investigated. All of those features were generated by decoding the APK manifest file, decompiling the dex code, and disassembling the applications' shared libraries. Our framework on the other hand focused mainly on permission request features.

Using APKtool [33-34] we created permissions request feature vectors and refined them by decoding the Android Manifest file which was used to develop the algorithm for permission request classification. Using permission features extracted from manifest files, our classification framework maximizes the permission components and generated a feature vector representation. Each of the feature vectors was discretely inputted into the Android Permission Classifier (APC) framework. There are many Android malware sample dataset repositories such as DREDIN [35], AndroZoo [36], Genome project [37], VirusShare [38]. Our framework performance was validated by a series of experimental results using 6528 Android malware samples obtained from Ashawa and Sarah [25]. Effective permission extraction accuracy and classification based on the threat level of such permission request and the corresponding protection level was achieved by our model.

The following contributions were made by our research:

1. We proposed a novel Android malware permission request classification framework that employs only permission features that replicates Android malicious application attributes.
2. We proposed a neural network technique for the generation of malicious feature vectors with effectual classification and representation of malicious binary features with mutual qualities with benign or non-threatening Android applications.
3. We offered a permission classification method tagged APC using neural networks, particularly deep learning. The method extracted all the permissions both malicious and benign application requests and then separated malicious requests from benign permissions.

4. We produced results that demonstrated our framework producing the best validation performance at the minimum epoch when compared with other classifiers. To the best of our knowledge, there has not been any work that used deep learning to classify Android malware permissions request based on their threat and protection levels.

The rest of the article is structured as follows. Section 6.2 provides a review of the literature by discussing related works. Section 6.3 presents the methods and the algorithmic approach. Section 6.4 presents the proposed framework description. Section 6.5 presents our evaluation procedure and experimental results, assessing the diversity capacity of our framework. Section 6.6 makes some concluding remarks and discusses future work.

6.2 Background and related work

While numerous research has been published on Android malware, a very limited amount of work has been done on malware protection and threat levels. In this section, related work on android malware similarity feature extraction and malware dynamic and static analysis were revisited. Then, we summarize and discuss permission requests and detection algorithms in the existing literature.

6.2.1 Similarity feature extraction

Similarity exists between non-zero vectors which might help when determining feature standardization. The study of Peters et al. [39] in a multimodal deep learning method for android malware detection; using various feature sets proposed a framework that used a similarity-based system to extract and represent android malicious application features. The binary vectors and feature set were calculated and represented as similarity features by creating a match with their record and characteristics existence using a similarity-based feature vector generation algorithm. The extracted features represented the reflective characteristics of an Android-based application.

Using a machine learning algorithm, the study of McLaughlin et al. [40] proposed an Android malware detection model that utilizes different Android Package Kit (APK) file features and permission requests information. Similar to the study of McLaughlin the research of Saxe and Berlin [41] applied classification and clustering algorithms to obtain permission that malicious application request on Android-based platforms. The study performed by David and Netanyahu [42] used

hardware and software features and components such as API calls, permission requests, and message intents to detect Android malicious applications. In addition, Kim [43] proposed a method whereby CFGs are used as features by utilizing keyword correlation gaps in FVG for (SVM) classification. The SVM imposes its features to have a weight that is uniformly distributed.

An investigation into the leakage of sensitive information by Android malicious applications, a data flow exploration was performed by TaintDraoid dynamically Sezer et al. [44]. Changes in the system' such as threads and processes were extracted and analysed. Abnormalities identified showed different usage of system features such as CPU by malicious applications. The research of Wang et al. [45] performed a systematic review on different mobile detection techniques approaches. The result obtained in the research identified detection techniques with their respective strengths and weaknesses. Other studies such as Yerima [46] proposed a model for detecting mobile usage abnormalities.

6.2.2 Permission requests and detection algorithms

Similarly, research [47] proposed a detection model where Convolutional Neural Network was used to detect malware on Android mobile devices. Sequences of opcode applications were used as features without alterations. Similarly, the study of Hao et al. [48] proposed a Deep Neural Network (DNN) framework for detecting malicious applications on Android-based devices. However, the method did not detect sophisticated and Zero-day malware that has similar attributes related to the benign application. Benign and malicious files were classified by the proposed work of Kim et al. [49] which uses Window dynamic API calls as feature vectors. The deep neural network-based model was believed to classify malicious files from benign files.

Related works that employed deep learning techniques include the research of Vasu and Pari [50] which used a multi-modal deep learning method, developed and an Android detection model using various features. The model learning strategy was based on a neural network. Using seven features namely String, opcode, API, shared library function, permission features, component, and environmental features; the model merged all the features into one feature and fed them into the classifier for malware detection. Their research did not in any given consideration to dangerous permissions request classification. The study of Pei et al. [51] using a Convolutional neural network proposed a detection technique for Android malware. The detection model used opcode features of android applications without modification.

Investigating the permissions with the risk that malicious applications use to explore Android resources, the study of Jha et al. [52] extracted permissions requests by Android applications using Random Forest. However, less relevant features that are inherent to an application were not explored. Though several previous works were carried out on identifying malicious applications using neural networks, none has investigated extracting and classifying permissions requests that android applications request using deep learning. In addition, none of the previous works considers the threat level that dangerous permission requests posed on the protection level of Android mobile devices. In the comparative results, the research identified the algorithms deployed in each of the techniques and their detection accuracy with their identified strengths and limitations, respectively.

The research intensively explored all the well-known detection techniques such as dynamic, static, hybrid, content-based, and emulation-based detection. However, they did not cover deep learning techniques for Android malware detection. What distinguished our work from the previous work is that though many of the previous research used several features and did not focus on classifying permission requests based on their threat and protection levels. Though several previous works were carried out, none has investigated the threat level that dangerous permission requests posed on the protection level of Android mobile devices. This formed part of the novelty of our research.

6.3 Materials and methods

6.3.1 Malware dataset

The Android malware dataset used for this research was obtained from Contagio [31] and DREBIN [46] Android malware dataset repository. Using an online VirusTotal scanner, we scanned to ascertain the benignity and maliciousness of the sample files. After scanning, the APK file was unzipped to extract the file contents of the dataset folder. The dataset consisted of 5560 malicious applications and 9476 benign application samples. The whole dataset relation consisted of 15 035 instances in an APK file format.

6.3.2 APK file decompilation

To obtain the resources contained in the raw dataset, the APK file was decompiled using Apktool embedded in the newest version of the Androguard. The decompiled APK contains lib, res, original assets, and the Android Manifest file application resources. The output of the decompiled APK generated Smali and Java class folders holding the resource files. Features such as manifest permissions, API call signatures, intents, and command signatures were obtained to form feature sets and input vectors to the network. Other features such as shared lib and libdata were also extracted after disassembling the source code of the .so file. The manifest permissions were extracted from the AndroidManifest.xml file extension while the API call signatures were extracted from Android class files. Using class conditional probabilities with minimum values, a total of 215 attribute partitions were extracted from pr_partition_0. The attributes extracted formed the general feature set. Specifically, the attributes contain 113 manifest permissions, 72 API call signatures, 23 intents, 6 command signatures, and binary attributes for (benign = B, malware = S [1]). In summary, our proposed classifier uses the following features: permission feature, API call feature, command feature, and intent feature, respectively.

6.3.2.1 Permission features

Permission manifest is one of the essential access control security mechanisms that Android uses. This access control mechanism (permission) places a constraint on the operational performance of any Android process. Permissions requested by applications provide relevant information regarding the application's behaviour. Permission features are usually defined or declared in the manifest file of the Android APK. During application installation, the manifest.xml file holds significant information about an application. When permission is granted, the application can be able to use and interact with the device-protected features. Android permissions are classified into four distinct threat levels namely, normal permission, dangerous permissions, signature permission, and signature/System permission. The study grouped both protection levels and manifest permissions as permissions features for the framework.

6.3.2.2 API call features

Android architecture is designed in such a way that the API of the system framework interacts with the Android system. Android API is made of two major components namely: classes, and packages. Malicious applications often invoke API. API invocational provides information on the application behaviour. Reverse engineering of the Android APK extracted all the API features in the dataset used for this research.

6.3.2.3 Command signature features

These are the command instruction sets for signing the Android label and activity class, respectively. Command signatures are used to sign requested permissions that applications request. In the case of benign applications, command signature signs applications whose signature certificate is the same as that of the permissions declared. When there is a match of certificates, Android automatically grants access to the application, without notifying the user or seeking his approval to grant access. Malicious applications that have the same signature as the declared permissions in the device protection level can easily have access to the system resources in the background. While permission request features of a malicious file are harmful to system information, command features are to the system's data and hardware.

6.3.2.4 Intent features

An Intent is the theoretical or abstract description of an action or activity to be implemented or performed. Android uses intent to launch and broadcast processes to any component of interest. Intents communicate with different services in the background; thus, facilitating the runtime binding of different application codes. Android uses intent features to launch activities. Intent contains two primary attributes, action and data. Other secondary attributes such as the category of the action to be performed on the data, component class to be used and, extras formed the metadata.

6.4 Feature selection and database generation

To improve the classifier's performance efficiency and reduce its computational cost, we reduced the number of the model's input variables. The selected input variables in this research were those whose correlation with the target variables was strongest. To achieve this, the feature ranking approach [53] was applied as a preliminary phase in determining the application characteristics and input variables reduction. Various android features such as the API call signatures, intents, manifest permissions, and command signatures were extracted from the APK. These features formed the general feature vectors. Features extracted formed the database of the feature vectors. In generating, the database for the feature vectors, the extracted features were classified into two different features based on their class pattern.

To normalize the feature value, the mix-max scaling approach [54] was applied by scaling feature values between the intervals of 0 to 1. Features with no value attributed to being scaled as zero (0) while those with values attributed to were scaled as one (1), respectively. Using the K-means clustering method, eighty (8) iterations were performed within-cluster sum of squared errors. C0 and C1 are the cluster labels for classes1 and 2 with counts 66 and 34, respectively. The class attribute has 66.0 weight in C0 and 34.0 weight in C1(as shown in Table 6-1). These formed the cluster centroids for the feature vector attributes of the dataset attributes distribution (as shown in Table 6-2).

A total of eight iterations was performed within-cluster sum of squared errors using the K-means clustering method. C0 and C1 are the cluster labels for classes 1 and 2 with counts 66 and 34, respectively. These formed the cluster centroids where permission attributes are distributed accordingly. Cluster attributes logical table contain the attribute class and the logical values (as shown in Table 6-3) while the decision list (as shown in Table 6-4) shows the associated rules formed from the attribute logical tables, respectively. All the feature categories were then combined to form a unified set of vectors which were used as the input vector. The input vector was converted into Comma Separated Values (CSV). The reason for converting the input vector into CVS format was to make it friendly with tools that find it difficult to accept datasets without a field separator. All the four feature categories were generated using K-means. Then combined to form a unified set of vectors which were used as the input vector. The input vector was converted into CSV.

Table 6-1 Cluster containing feature vector binary attributes

Clusters	Binary features
Cluster 0 (c0): S	0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0, 1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,0,0,0,S
Cluster 1 (c1): B	1,1,1,1,1,1,0,1,1,1,1,0,0,0,1,1,1,1,0,1,0,0,1,0,0,0,1,1,0,0,1,1, 1,0,0,1,0,0,0,0,0,0,1,0,1,0,1,0,0,1,1,0,1,0,0,1,1,0,0,0,0,1,0,0, 1,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,1, 0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, 0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, 0,0,1,0,0,1,1,1,0,0,0,B

Table 6-2 Data set attributes distribution

Statistical attributes	Distinct values
Minimum	0
Maximum	1
Mean	0.426
StdDev	0.495
Class pattern	8624
Benign	[0, 0.071]
Class pattern	6412
Malicious	[0.929, 1]

The qualitative characteristics of malware dataset, Distinct values: These are calculated values held by each statistical attribute, Minimum: 0 and 1 are the minimum and maximum binary values contained in each cluster (Cluster 0, Cluster 1), respectively, Mean: statistical mean of the dataset distribution, StdDev: SD. The reason for converting the input vector into CVS format was to make

it friendly with tools that find it difficult to accept datasets without a field separator. The algorithm for feature vector generation (as shown in Algorithm 1).

Algorithm 1 Generation of Combined Feature Vectors from the Binary Feature Vector Database

Input: Android applications' feature set contained in the Database

Output: Combined feature vectors

```

1: decompile APK // to access all the APK components
2: set_partition_index_feature_vector ← [1|0|0|1|0|...0|1]
3: centroids ← K_means
4: for index ← 0 // setting the feature vector index
5: for  $\forall \mathcal{F}s \in \beta\mathcal{FV}_{db}$  do while
6: if  $\mathcal{F}s \in \mathcal{FV}_{Android\_app}$ 
7: then
8: combined_feature_vectors[index] ← 1
9: return combined_feature_vectors
10: exit

```

Table 6-3 Attributes logical table

Attribute	Logical values
@attribute a0	{false,true}
@attribute a1	{false,true}
@attribute a2	{false,true}
@attribute a3	{false,true}
@attribute a4	{false,true}
@attribute a5	{false,true}
@attribute a6	{false,true}
@attribute a7	{false,true}
@attribute a8	{false,true}
@attribute a9	{false,true}
@attribute class	{c0,c1}

The feature vector information is extracted from the dataset from the malicious feature database which consists of strings, permissions, components, and other environmental features. The structure of the database was composed of different components where all the features were pruned and classified. All the string features with their respect hashes and sizes, permission features are in <uses-

permission> and < permission> tag. All:<provider>, <filter>, <activity>, <intent>, <grant-uri permission>, <service>, and < receiver> formed the general component of the feature database where all the distinct features are extracted and classified. The environmental components: <library> and < uses-sdk>. Opcode and API methods: send_message, socket, and invoke. Details of the feature vector set information extracted by the algorithm which contained intent (as shown in Table 6-5), commands signature (as shown in Table 6-6), permission (as shown in Table 6-7), and API call signature (as shown in Table 6-8) formed the major feature categories of the combined database, respectively.

6.5 Feature classification

The approach used for the feature classification process in this research is the backpropagation mechanism [55]. Back-propagation constitutes the central learning hub for producing an accurate classification outcome. It is the information transmission with the error which neural network generates when trying to make a prediction or classification about a given dataset. This is composed of the iteration and learning layers (input, hidden, and output layer) with distinctive random initialization weights and biases. Our model applied the information gain approach [56] as a feature classification methodology. The *IG* processes the amount of information a feature *X* provides about the class. Let *X* be a feature in a given Android application sample containing *x_i* possible outcomes, the *IG* is represented as follows:

$$IG = H(X) - H\left(\frac{x}{y}\right) \quad (6-1)$$

The $H(X)$ from the above equation is the measure of randomness in the processed information which forms the entropy of feature *X*, while $H\left(\frac{x}{y}\right)$ is the entropy after observing *Y* features respectively. Given the probability density function (PDF) $p(x)$ for random variable *X*, the $H(X)$ is defined as:

$$H(X) = -\sum p(x) \log_2(p(x)) \quad (6-2)$$

Similarly, the $H(X)$ of *X* comparative to feature *Y* expressed as:

$$H\left(\frac{x}{y}\right) = -\sum p\left(\frac{x}{y}\right) \log_2 \left(p\left(\frac{x}{y}\right)\right) \quad (6-3)$$

The function $p\left(\frac{x}{y}\right)$ is the conditional probability. High reduction of the selective information in X increases the significance of its features at the peak of density function when the conditional probability in the sample pattern of both malicious and benign features increases. Thus, the selective information in X decreases as potential difference function decreases, respectively. We assumed that there was no probabilistic failure that existed at any time (t) during the attribute's distribution and clustering.

Table 6-4 Decision list using associated rules

Rule	Decision List
RULE 0:	c0: = not(a3), not(a2), a6, a0
RULE 1:	c1: = not(a9), a2, not(a4), not(a1), a5
RULE 2:	c0: = a3, a4, a6, not(a9), not(a5)
RULE 3:	c1: = a1, a6, a0, not(a7), a3, a9, not(a5),
RULE 4:	c0: = not(a5)
RULE 5:	c1: = a4, not(a3), a2, not(a8), a9, not(a0),
RULE 6:	c0: = not(a2), a3, a7, not(a4), not(a6), a1,
RULE 7:	c1: = not(a8), a5, a9
RULE 8:	c0: = a6, not(a3), not(a4)
RULE 9:	c1: = a5, not(a6), not(a2), not(a9),
RULE 10:	c0: = a5, not(a4), not(a2)
RULE 11:	c1: = not(a2), not(a6), a7, a8, a3, not(a0),
RULE 12:	c0: = a9, not(a3), a1, a0, a5
RULE 13:	c1: = not(a9), a6, a7, a4, a2, a3, not(a1),
RULE 14:	c0: = a6, a0, a5, a8, a4, a7, a3
RULE 15:	c1: = not(a9), a0, not(a8), a6, a5, a4, a1
RULE 16:	c0: = a1, a3, a5, a9, a2
RULE 17:	c1: = not(a4)
RULE 18:	c0: = a2
RULE 19:	c1: = not(a0), not(a6), a3, a8, a7, a4, a9
RULE 20:	c0: = not(a0), a8, not(a2), not(a9), a5

On the other hand, if a probability failure occurs at any given survival; the risk function $h(t)$ will occur by forming an integral and instantaneous ratio. The hazard or risk function $h(t)$ was defined as follows:

$$H(x) = \int_{\infty}^x h(t)dt \quad (6-4)$$

where ∞ is the range of time t from 0 to ∞ . Our research assumed that all the features in the dataset survive the past time and hazard function. Thus, the survival function was defined as:

$$s(t) = p_r(T > t) = 1 - F(t) \quad (6-5)$$

where $S(t)$ is the survival function at time t , T is the random variable, P_r is the probability that some feature survived, and F is the event density. Using three different distributions: Weibull, normal, and Birnbaum–Saunders, we demonstrated that at each intersection of either of the distributions, as F increases, the corresponding value of $S(t)$ decreases, respectively. This provides a guide in studying the binary vectors in the dataset during feature representation.

6.6 Binary feature representation

In this subsection, we described the permission classification problem of our proposed classifier. Our binary feature vectors were obtained by performing eight iterations within-cluster sum of squared errors using the K-means clustering method. Let m represent a set of given G programs given as $G = \{g_i \dots g_m\}$ with $g_i \in G$. Let the class label C_i and the feature vector f_i be matched as $g_i \in G$ for every $g_i \in G$.

We defined our class label and feature vector as follows:

$$C_i \in \{c_m, c_b, c_k\} \quad (6-6)$$

$$f_i \in \{f_1, f_2, \dots, f_k\} \quad (6-7)$$

where C_m , C_b , C_k represents class label for the malicious request, class label for the benign permission request, and class label for an unknown permission request respectably. Let $R = \{r_1, r_2, \dots, r_n\}$ represent n classifiers set that label permission requests from the input APK file as either benign or malicious. Let $r_i = \{a_i, C_i, T_i\}$, where T_i is the classifiers' computational time, a_i is the classification accuracy, and C_i is the function that labels and maps the feature vector f to c and a_i , respectively, upon the training set represented as $C_i(f): f \rightarrow c$.

Table 6-5 Intent feature vectors

Feature	Category
android.intent.action.BOOT_COMPLETED	Intent
android.intent.action.PACKAGE_REPLACED	Intent
android.intent.action.SEND_MULTIPLE	Intent
android.intent.action.TIME_SET	Intent
android.intent.action.PACKAGE_REMOVED	Intent
android.intent.action.TIMEZONE_CHANGED	Intent
android.intent.action.ACTION_POWER_DISCONNECTED	Intent
android.intent.action.PACKAGE_ADDED	Intent
android.intent.action.ACTION_SHUTDOWN	Intent
android.intent.action.PACKAGE_DATA_CLEARED	Intent
android.intent.action.PACKAGE_CHANGED	Intent
android.intent.action.NEW_OUTGOING_CALL	Intent
android.intent.action.SENDTO	Intent
android.intent.action.CALL	Intent
android.intent.action.SCREEN_ON	Intent
android.intent.action.BATTERY_OKAY	Intent
android.intent.action.PACKAGE_RESTARTED	Intent
android.intent.action.CALL_BUTTON	Intent
android.intent.action.SCREEN_OFF	Intent
intent.action.RUN	Intent
android.intent.action.SET_WALLPAPER	Intent
android.intent.action.ACTION_POWER_CONNECTED	Intent
android.intent.action.BATTERY_LOW	Intent

Table 6-6 Commands signature feature vectors

Feature	Category
mount	Commands signature
chmod	Commands signature
remount	Commands signature
chown	Commands signature
/system/bin	Commands signature
/system/app	Commands signature

Table 6-7 Permission feature vectors

Feature	Category	Feature	Category
SEND_SMS	Manifest Permission	WRITE_SECURE_SETTINGS	Manifest Permission
READ_PHONE_STATE	Manifest Permission	DUMP	Manifest Permission
GET_ACCOUNTS	Manifest Permission	BATTERY_STATS	Manifest Permission
RECEIVE_SMS	Manifest Permission	ACCESS_COARSE_LOCATION	Manifest Permission
READ_SMS	Manifest Permission	SET_TIME	Manifest Permission
USE_CREDENTIALS	Manifest Permission	WRITE_SOCIAL_STREAM	Manifest Permission
MANAGE_ACCOUNTS	Manifest Permission	WRITE_SETTINGS	Manifest Permission
WRITE_SMS	Manifest Permission	REBOOT	Manifest Permission
READ_SYNC_SETTINGS	Manifest Permission	BLUETOOTH_ADMIN	Manifest Permission
AUTHENTICATE_ACCOUNTS	Manifest Permission	BIND_DEVICE_ADMIN	Manifest Permission
WRITE_HISTORY_BOOKMARKS	Manifest Permission	WRITE_GSERVICES	Manifest Permission
INSTALL_PACKAGES	Manifest Permission	KILL_BACKGROUND_PROCESSES	Manifest Permission
CAMERA	Manifest Permission	STATUS_BAR	Manifest Permission
WRITE_SYNC_SETTINGS	Manifest Permission	PERSISTENT_ACTIVITY	Manifest Permission
READ_HISTORY_BOOKMARKS	Manifest Permission	CHANGE_NETWORK_STATE	Manifest Permission
INTERNET	Manifest Permission	RECEIVE_MMS	Manifest Permission
RECORD_AUDIO	Manifest Permission	SET_TIME_ZONE	Manifest Permission
NFC	Manifest Permission	CONTROL_LOCATION_UPDATES	Manifest Permission
ACCESS_LOCATION_EXTRA_SETTINGS	Manifest Permission	BROADCAST_WAP_PUSH	Manifest Permission
WRITE_APN_SETTINGS	Manifest Permission	BIND_ACCESSIBILITY_SERVICES	Manifest Permission
BIND_REMOTEVIEWS	Manifest Permission	ADD_VOICEMAIL	Manifest Permission
READ_PROFILE	Manifest Permission	CALL_PHONE	Manifest Permission
MODIFY_AUDIO_SETTINGS	Manifest Permission	BIND_APPWIDGET	Manifest Permission
READ_SYNC_STATS	Manifest Permission	FLASHLIGHT	Manifest Permission
BROADCAST_STICKY	Manifest Permission	READ_LOGS	Manifest Permission
WAKE_LOCK	Manifest Permission	SET_PROCESS_LIMIT	Manifest Permission

RECEIVE_BOOT_COMPLETED	Manifest Permission	MOUNT_UNMOUNT_FILESYS	Manifest Permission
RESTART_PACKAGES	Manifest Permission	BIND_TEXT_SERVICE	Manifest Permission
BLUETOOTH	Manifest	INSTALL_LOCATION_PROVIDE	Manifest Permission
READ_CALENDAR	Manifest	SYSTEM_ALERT_WINDOW	Manifest Permission
READ_CALL_LOG	Manifest	MOUNT_FORMAT_FILESYSTEM	Manifest Permission
SUBSCRIBED_FEEDS_WRITE	Manifest	CHANGE_CONFIGURATION	Manifest Permission
READ_EXTERNAL_STORAGE	Manifest	CLEAR_APP_USER_DATA	Manifest Permission
VIBRATE	Manifest	CHANGE_WIFI_STATE	Manifest Permission
ACCESS_NETWORK_STATE	Manifest	READ_FRAME_BUFFER	Manifest Permission
SUBSCRIBED_FEEDS_READ	Manifest	ACCESS_SURFACE_FLINGER	Manifest Permission
CHANGE_WIFI_MULTICAST_S	Manifest	BROADCAST_SMS	Manifest Permission
WRITE_CALENDAR	Manifest	EXPAND_STATUS_BAR	Manifest Permission
MASTER_CLEAR	Manifest	INTERNAL_SYSTEM_WINDOW	Manifest Permission
UPDATE_DEVICE_STATS	Manifest	SET_ACTIVITY_WATCHER	Manifest Permission
WRITE_CALL_LOG	Manifest	WRITE_CONTACTS	Manifest Permission
DELETE_PACKAGES	Manifest	BIND_VPN_SERVICE	Manifest Permission
GET_TASKS	Manifest	DISABLE_KEYGUARD	Manifest Permission
GLOBAL_SEARCH	Manifest	ACCESS_MOCK_LOCATION	Manifest Permission
DELETE_CACHE_FILES	Manifest	GET_PACKAGE_SIZE	Manifest Permission
WRITE_USER_DICTIONARY	Manifest	MODIFY_PHONE_STATE	Manifest Permission
REORDER_TASKS	Manifest	CHANGE_COMPONENT_ENABL	Manifest Permission
WRITE_PROFILE	Manifest	CLEAR_APP_CACHE	Manifest Permission
SET_WALLPAPER	Manifest	SET_ORIENTATION	Manifest Permission
FEATURESBIND_INPUT_METH	Manifest	READ_CONTACTS	Manifest Permission
READ_SOCIAL_STREAM	Manifest	DEVICE_POWER	Manifest Permission
READ_USER_DICTIONARY	Manifest	HARDWARE_TEST	Manifest Permission
PROCESS_OUTGOING_CALLS	Manifest	ACCESS_WIFI_STATE	Manifest Permission
CALL_PRIVILEGED	Manifest	WRITE_EXTERNAL_STORAGE	Manifest Permission
BIND_WALLPAPER	Manifest	ACCESS_FINE_LOCATION	Manifest Permission
RECEIVE_WAP_PUSH	Manifest	SET_WALLPAPER_HINTS	Manifest Permission

Table 6-8 API call signature feature vectors

Feature	Category	Feature	Category
onServiceConnected	API call signature	createSubprocess	API call signature
bindService	API call signature	URLClassLoader	API call signature
attachInterface	API call signature	abortBroadcast	API call signature
ServiceConnection	API call signature	TelephonyManager.getDeviceId	API call signature
android.os.Binder	API call signature	getCallingPid	API call signature
Ljava.lang.Class.getCanonicalName	API call signature	Ljava.lang.Class.getPackage	API call signature
Ljava.lang.Class.getMethods	API call signature	Ljava.lang.Class.getDeclaredClasses	API call signature
Ljava.lang.Class.cast	API call signature	PathClassLoader	API call signature
Ljava.net.URLDecoder	API call signature	TelephonyManager.getSimSerialNumbe	API call signature
android.content.pm.Signature	API call signature	Runtime.load	API call signature
android.telephony.SmsManager	API call signature	TelephonyManager.getCallState	API call signature
getBinder	API call signature	TelephonyManager.getSimCountryIso	API call signature
ClassLoader	API call signature	sendMultipartTextMessage	API call signature
Landroid.content.Context.registerReceiver	API call signature	PackageInstaller	API call signature
Ljava.lang.Class.getField	API call signature	sendDataMessage	API call signature
Landroid.content.Context.unregisterReceiver	API call signature	HttpPost.init	API call signature
Ljava.lang.Class.getDeclaredField	API call signature	Ljava.lang.Class.getClasses	API call signature
getCallingUid	API call signature	TelephonyManager.isNetworkRoaming	API call signature
Ljavax.crypto.spec.SecretKeySpec	API call signature	HttpRequest	API call signature
android.content.pm.PackageInfo	API call signature	divideMessage	API call signature
KeySpec	API call signature	Runtime.exec	API call signature
TelephonyManager.getLine1Number	API call signature	TelephonyManager.getNetworkOperato	API call signature
DexClassLoader	API call signature	MessengerService	API call signature
HttpGet.init	API call signature	IRemoteService	API call signature
SecretKey	API call signature	Set_Alarm	API call signature
Ljava.lang.Class.getMethod	API call signature	Account_Manager	API call signature
System.loadLibrary	API call signature	TelephonyManager.getSimOperator	API call signature
android.intent.action.SEND	API call signature	onBind	API call signature
Ljavax.crypto.Cipher	API call signature	Process.start	API call signature
android.telephony.gsm.SmsManager	API call signature	Context.bindService	API call signature
TelephonyManager.getSubscriberId	API call signature	ProcessBuilder	API call signature
Runtime.getRuntime	API call signature	Ljava.lang.Class.getResource	API call signature
Ljava.lang.Object.getClass	API call signature	defineClass	API call signature
Ljava.lang.Class.forName	API call signature	findClass	API call signature
Binder	API call signature	Runtime.loadLibrary	API call signature
android.os.IBinder	API call signature	transact	API call signature

The model computational time (TR) and the classification rate of accuracy (AR) is defined by the following equations, respectively:

$$TR = \sum_{i=1}^n T(r_i) | r_i \in R \quad (6-8)$$

$$AR = \sum_{i=1}^n a(r_i) | r_i \in R \quad (6-9)$$

To prevent the classifiers from going beyond the threshold level (φ) given that $AR \leq \varphi$, and all the n classifiers in our model must not exceed the minimum classifiers' number (E), we defined the minimum function in the system operates as follows:

$$\min_n = \sum_{i=1}^n r_i \leq E \quad (6-10)$$

Given a condition that each permission requested by the application is labelled as either malicious or benign as $g_i \in \{C_m, C_b\}$, $\forall g_i \in G$, $1 \leq i \leq m$; where C_m represents class label for the malicious permission request, C_b represents class label for the benign permission request, respectively. We included the classifiers' number E to help determine the effectiveness of our classifier. It also helps comparison with different machine learning classifiers in case there is a disparity in classification performance between them. Each binary number represents a cluster attribute in the decision list. Using 2-fold cross-validation, we constructed attributes rules with each cluster in the decision list. Concerning the feature vectors, each attribute can either be a false or a true and must contain each value representation in both clusters (cluster 0 and cluster 1). All the features were spread over `pr_partition_0` which shows the distribution value for each attribute in a density-based clustering pattern. The cluster attributes from attribute `a0` to attribute `a9` were formed using association rules [57] for feature mining attributes logical values (see Algorithm 2).

Algorithm 2 Classification of Permission Feature Vector from the Database

Input: Android applications' manifest xml file

Output: Android permission features

```
1: all permissions  $\leftarrow \forall \delta$  // generating permission information
2: for 0 to Max_iteration do
3: if all perm ==  $\delta$ 
4: then for 0 to total perm do
5: malicious perm  $\leftarrow$  select feature (0, 1) // padded zeros and leading
   zeros
6: newperm  $\leftarrow$  newperm  $\cup$  {malicious}
7: else
8: for new.total () < totalperm do
9: select K, L  $\in$  (0, |allpermissions|)
10: malicious = K_crossover (allpermissions[k], allpermissions[l])
11: for maliciousperm  $\in$  newperm do
12: for benignpermk  $\in$  newperm do
13: if  $\exists$  maliciousperm | maliciousperml  $\in$  newperm  $\wedge$  k  $\neq$ 
    l.maliciouspermk > maliciousperml
14: then
15: newperm  $\leftarrow$  newperm \ maliciouspermk
16: if newperm  $\sqsubset \forall$  maliciousperm then
17: permission_feature_vector [index]  $\leftarrow$  1
18: else
19: permission_feature_vector [index]  $\leftarrow$  0
20: return android_permission_feature_vectors
21: exit
```

6.7 Framework description

This subsection of the paper describes the neural network approach applied in our proposed framework known as Android Permission Classifier (APC). The principle used in developing our classification model is the principle of convolutional neural networks for recognizing the pattern in the dataset [58-59]. We defined our neural network system $s \in \{1, 2, 3\}$ to consist of l number of layers defined as $l^s = \{1, 2, \dots, n\}$ respectively. In our framework, the vector input x that enters the initial network was represented as x^s . From linear transformation, the output from the initial network was transformed from a matrix to another vector (Z) into the second layer to form the incoming vector represented as $Z^{(x^s)}$, where Z is the transformation function. From partial differentiation, $y = f(x)$ where y is the output of the function. The rectified linear unit (ReLU) as the activation function applied in our neural network model is defined as:

$$y = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases} \quad (6-11)$$

The transformation z over the given input vector x , weights w and biases b on the above function is defined as:

$$z = f(x, w, b) \quad (6-12)$$

$$y = g(x) \quad (6-13)$$

To design a classification model that captures multidimensional data inputs with weights and biases, we applied max-pooling operation after convolution with Relu function as follows:

$$z = \tanh(f(x_1, w_1, b_1,)) \quad (6-14)$$

$$z = \text{Relu}(f(z, w_2, b_2,)) \quad (6-15)$$

$$\text{Relu}(x) = \max(f(0, z)) \quad (6-16)$$

Where ReLu is the activation function of the network, x is the input vector, w is the network weight and b is the biases in the network respectively. The output, weights, and biases from the network layer l^s while applying the ReLu function (activation function) in the initial network using backpropagation was defined by the following equations:

$$y(l^s + 1) = f(z(l^s + 1)) \quad (6-17)$$

$$z(l^s + 1) = w(l^s + 1).y(l^s) + b(l^s + 1) \quad (6-18)$$

where W are the network weights, b are the network biases, $y(l^s)$ output vector and f is the network activation function, respectively. Based on the working principle illustrated in the transformation operation shown in the figure above, the output in the initial network layer was transformed and became an input vector in the final network of our neural network model as follows:

$$z'(^{(l'+1)}) = w'(^{(l'+1)}).y'(^{(l^s)} + b'(^{(l^s+1)}) \quad (6-19)$$

$$y'(^{(l'+1)}) = Relu(z'(^{(l'+1)})) \quad (6-20)$$

To help predict the probability of the output between 0 and 1 range, squashing function (σ) was applied at the last layer of the network. This is to determine permission is malicious (1) or benign (0) in our framework output. Given the quashing function σ , the number of layer n in the neural network, the output vector of the final layer y' , the input vector x in the network. The transformation at the last layer of the network was defined as follows:

$$y'^{(n_x)} = \sigma(z'^{(n_x)}) \quad (6-21)$$

Where σ is the applied squashing function, n is the number of layers in the neural network, y' is the output vector of the final layer, x is the input vector in the network, and Z is the matrix-vector respectively. The overall system architecture (as shown in Figure 6-1) illustrates the various processes performed from dataset collection to binary feature vector generation.

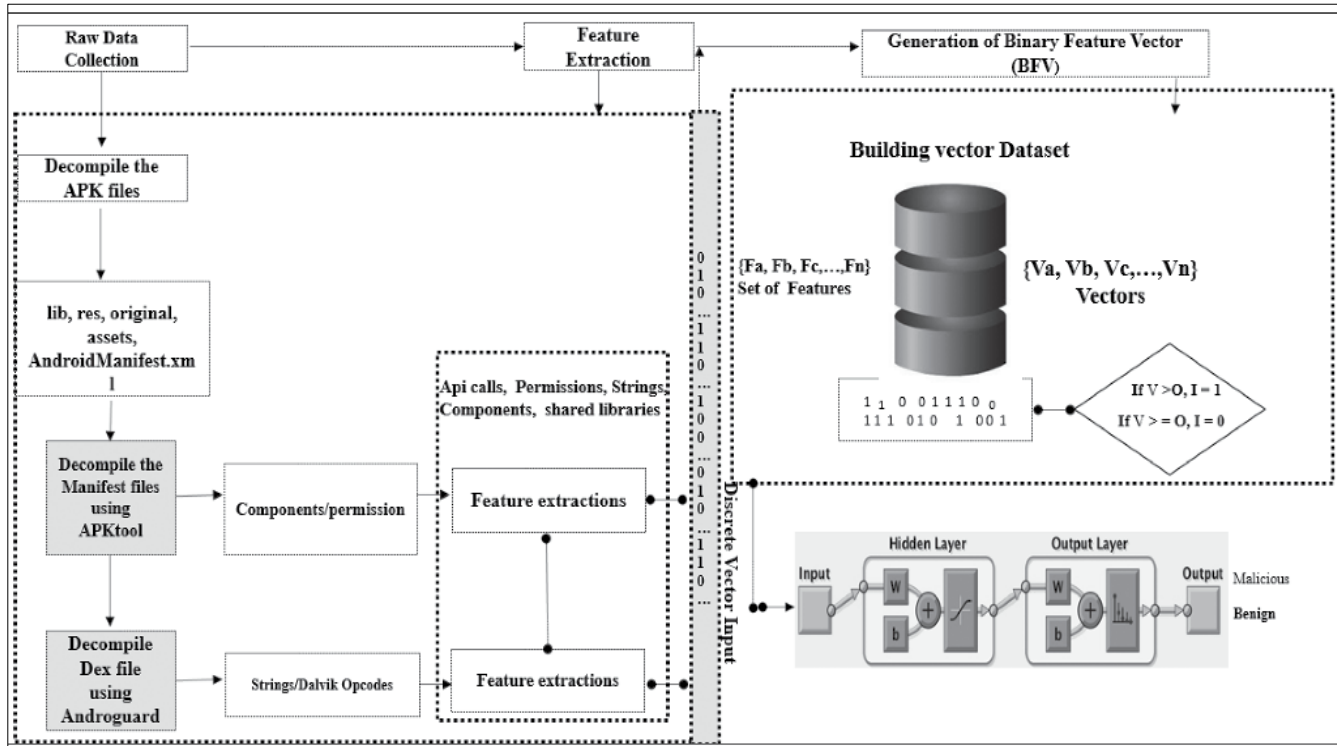


Figure 6-1 The overall system architecture

6.7.1 Network overfitting

When developing a classification model using a neural network, network overfitting characterizes the basic drawback in many models. To avoid overfitting occurrences, our research applied the Bayesian regularization (BR) technique [60-61]. The choice to choose the Bayesian regularization over other algorithms such as Levenberg–Marquardt is its capability to show potential convoluted relationships between variables in the distribution of a dataset. Therefore, providing robustness in the model. Unlike Dropout regulation [62] that skips some units randomly during training of the network, the Bayesian regularization technique skips the model from bias definition [63], thus, making the model reliant on biases and weights associated with the network units. Using BR in the APC model provides the framework with enhanced generation performance and achieved the best network validation performance. The overall problem definition of the Bayesian regularization training to overcome network overfitting is defined in the equation as follows:

$$y(t) = f(t(t - 1), \dots, x(t - d), y(t - 1), \dots, y(t - d)) \quad (6-22)$$

Where x is the random variable taken definite values between 0 and 1 in the dataset of X independent variables, d is the number of delays in the network, t is the target variable, f is the nonlinear function of x variables, and $y(t)$ is the output of the target value. Our dataset was trained by introducing a linear function w to minimize the error in t . The expectation (E) of the non-linear function f is defined as:

$$E[f] = \int p(x).f(x)dx \quad (6-23)$$

We defined the average expected loss L when malicious permission is wrongly classified as benign as follows:

$$E[L] = \iint L(t, y(x))p(x, t)dxdt \quad (6-24)$$

Where $L(t, y(x)) = (y(x) - t)^2$ and $p(x, t)$ are normal distribution parameters. The MSE over the parameterised function is the model cost function. The neural network operation (as shown in Figure 6-2) schematically shows the operation function of the proposed classifier.

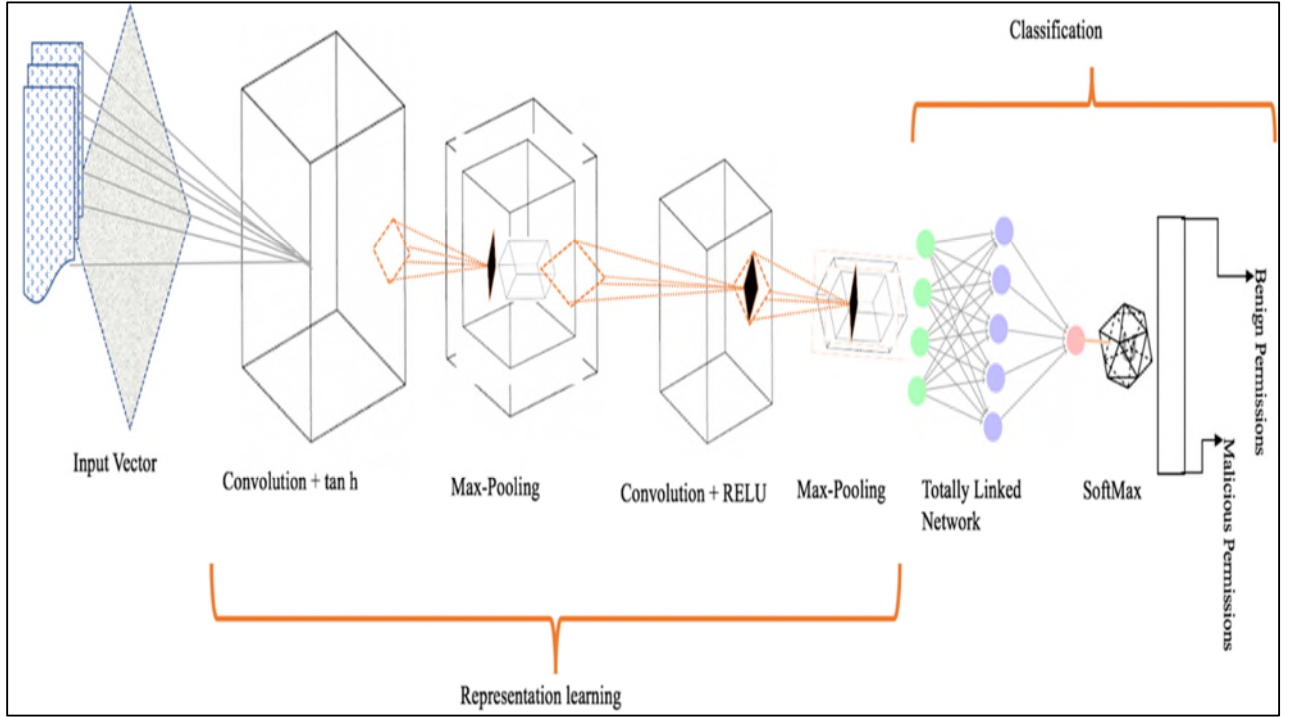


Figure 6-2 7An illustration of the neural network working operation function of the proposed classifier

6.7.2 Regularization of feature diversity

Feature redundancy (FRC) is one of the major factors that enhance errors in classification and detection frameworks. Most conventional classification algorithms are limited to inaccurate discrimination of features related to malware. This is due to the daily emergence of different malware variants with unique attributes. To minimise classification errors that could occur because of feature redundancy, the diversity regularization technique [64] was applied in our framework. During this process, we discarded all the features with self-correlation and only those whose correlation values are above the diversity threshold τ were considered. The diversity threshold provides a trade-off parameter to ensure that feature vectors in the malware dataset are adequately differentiated. Feature sharing across several high-level attributes is ensured.

(6-25)

$$D_r C^{(l)} f = \frac{1}{2} \sum_{i=1}^{n'} \sum_{j=1, i \neq j}^{n'} m_{i,j}^{(l)} (SIM_c(f_i, f_j)) (f_i, f_j)^{(l)}{}^2$$

Consider a vector space $V = \{f_1, f_2, \dots, f_n\}$, where f_s are feature vectors. The similarity that exists between two feature vectors say, f_1 and f_2 is determined by the correlation that occurs between f_1 & f_2 . During the training process, SIM_c is set to take definite values within the range of -1 to 1.

The SIM_c value is negative (-1) when f_1 & f_2 have opposite direction in the feature vector subspace V . This does not connote the vectors in this direction are negative feature vectors. SIM_c value is set to zero (0) if f_1 & f_2 in the vector space are in an orthogonal projection direction. For instance, when f_1 & f_2 are in orthogonal projection, we defined it in a bilinear form as follows: $\langle f_1 \& f_2 \rangle = 0$ for $i \neq j$. The regularization cost for feature vector diversity is abbreviated as (D_rC) in the APC neural network layer l^{th} with connecting feature networks given n number of neurons in l layer, and the binary mask variable $m_{i,j}$.

To determine the earliest time that features classification failure may occur, the threshold parameter τ was taken as a hyperparameter assuming values between 0 and 1. During the experiment, we avoided setting the hyperparameter to zero (0) to avoid feature orthogonality. This was considered due to feature-sharing abilities that exist in android-based applications. To enforce feature diversity during training, the diversity factor λ in the experiments was performed at 1, 5, 15, and 30 respectively using 100 iterations. The threshold parameter was performed at $\tau=0.1, 0.2$, and 0.3 respectively. The regularization cost for feature vector diversity is abbreviated as (D_rC) in the APC neural network layer l^{th} with connecting feature networks given n number of neurons in l layer, and the binary mask variable $m_{i,j}$, edged with the SIM_c and the features in the vector space is defined as:

The feature variables with binary mask $m_{i,j}$ and the hyperparameter $0 \leq \tau \leq 1$ with the dot product determinant (Ω) of i and j with the regularized layers defined as:

$$D_r M_{i,j} = \begin{cases} 1 & \tau \leq |\Omega_{i,j}| \leq 1 \\ 0 & i = j \\ 0 & otherwise \end{cases} \quad (6-26)$$

In equation (6-26) above, τ represent the threshold parameter of the regularised network. The similarity that exists between two feature vectors say, f_1 and f_2 , is determined by the correlation that occurs between f_1 & f_2 . This correlation is defined by the cosine similarity measure (SIM_c) function as:

$$SIM_c(f_1, f_2) = \frac{\langle f_1, f_2 \rangle}{\langle \|f_1\| \|f_2\| \rangle} \quad (6-27)$$

The overall D_rC of the FRc was evaluated to 2.413 which has a higher feature divergence capacity out of threefold measure.

6.7.3 System learning approach

To enhance the classifier's accuracy and to control the partial feature extraction problem, each permission type from the feature vector was learned from each initial layer of the neural network with a learning rate of 0.01. This approach is repeated from the initial layer to the final layer. At each layer, the succeeding layer learns from the preceding layer. Given the network layer $l^s = \{1, 2, \dots, n\}$, l^0 learns from l^1 , l^2 learns from l^1 respectively while updating each network. In addition, where a permission feature cannot be extracted in the manifest file, zeros were padded to form a feature vector whose attribute is not associated with permission feature vectors. This helps the classifier in detecting and differentiating permission features with a high threat to the protection level provided there is at least one (1) in the feature vector.

The learning strategy of the APC model showed a significant reduction in the time taken for the network to be trained. The approach was applied to overcome the more time it requires when using BR regularization algorithm. Though the Levenberg-Marquardt algorithm typically requires less time, it does not result in a good generalization for difficult, small, or noisy data. Apart from enhancing the classification accuracy of the model, produced a low MSE and SSE but it showed a high correlation coefficient and classification accuracy. Error estimation was significantly minimised using the approach due to the objection function in BR.

6.8 Experimental evaluation

In this section, we present the performance evaluation of the Android Permission Classifier (APC) framework. APC is for classifying Android permissions that malware requests to access Android device resources. Different experiments were conducted to determine the classifier's performance. This section also presented the experimental design, the classifier's performance metric, comparison with state-of-art classification systems, discussion, and conclusions.

6.8.1 Experimental setup

We experimented in a controlled environment using a VMware workstation. The virtual environment was created and used for this research to avoid malware escalation to the outside world. The virtualization approach of the research experimental design helped in ensuring that the network infrastructure and other system resources are not affected. Our framework was implemented in python, Keras library [65], TensorFlow [66], and Scikit learn [67] were used for modelling our neural network, cluster modelling, and other algorithms for the classifier. We analysed the result and performance of our APC model using the WEKA toolkit [68]. WEKA is an open-source Waikato environment for knowledge analysis.

6.8.2 Evaluating performance metric

To evaluate the classifier performance, Area Under Curve (AUC) was used to measure the classifier's performance. This measurement approach helps to evaluate how efficiently the classifier can classify permission requests that have been benign or malicious. Also, AUC been the area under the Receiver Operating Characteristics (ROC) curve displays the graphical representation or illustration of the binary classifier's analytical ability for its discriminate threshold (as shown in Figure 6-3). A different experimental test was used to derive variance measures. Precisely, our research used threshold averaging ROC curve methodology [69] rather than vertical averaging. Although vertical averaging [70] easily makes single averages of dependent variables and TPR which significantly calculates the confidence intervals and makes it simplified.

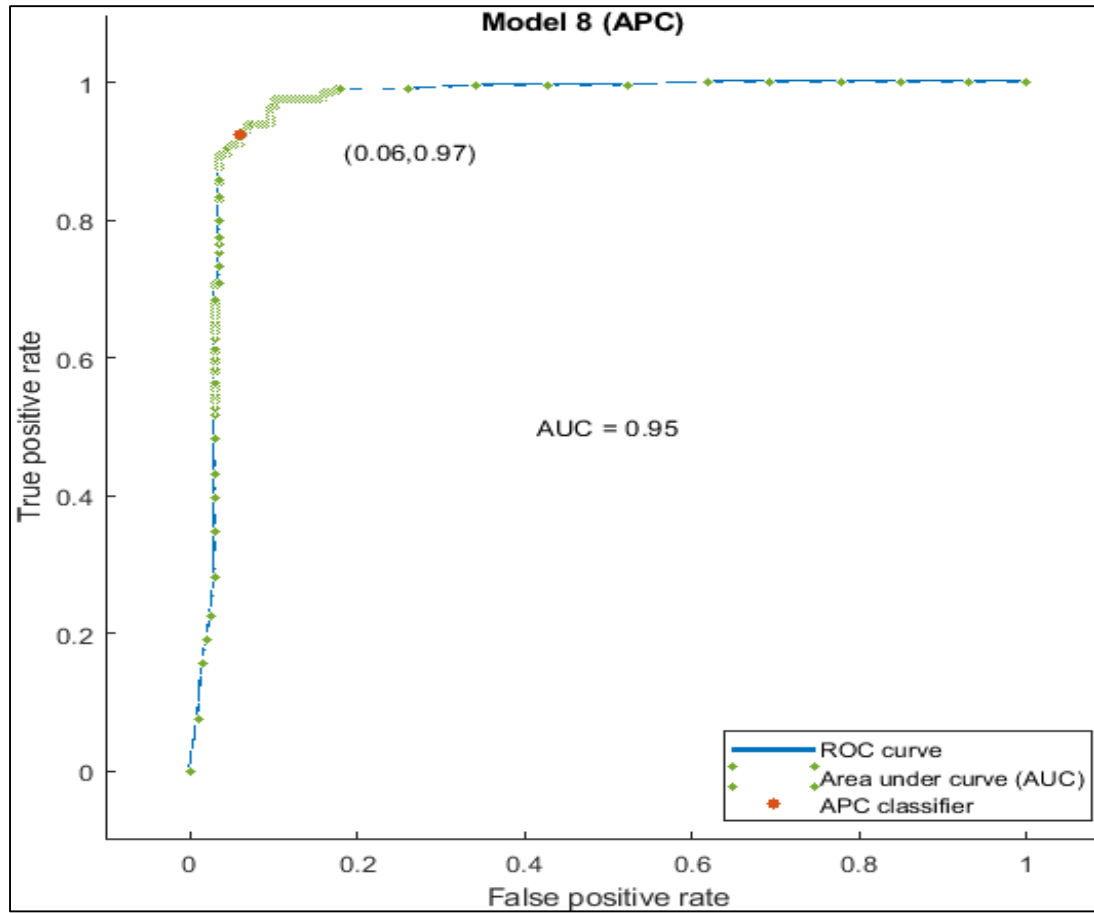


Figure 6-3 ROC curve showing the AUC of the classifier

However, when using a vertical averaging approach, the researcher does not have direct and absolute control of the FPR and independent variables. As a result, we used a threshold averaging curve to control the classifier's threshold scores by ensuring that at each threshold, the corresponding ROC curve was averaged. The model performance metrics when compared with existing models (see Table 6-9) and traditional classifiers (see Table 6-10) show the highest precision and accuracy. Also, when compared with traditional classifiers, APC shows the highest capacity diversity in feature classification (see Figure 6-4). The detail of the feature diversity is discussed in section 6.8.3 of this chapter.

Table 6-9 Performance metrics comparison [34][71][72]

Model	Precision (%)	Recall (%)	FPR (%)	F1-Score (%)	ACC (%)	AUC (%)
APC	97.0	98.60	0.03	98.0	97.00	95.00
[34]	94.30	94.50	0.06	94.60	94.38	94.60
[71]	91.60	90.70	0.08	91.80	90.72	86.00
[72]	92.40	87.50	0.08	94.12	90.54	92.00

The APC framework used the AUC also called the c-statistic of the ROC curve for performance evaluation of the APC classifier. An AUC value of >0.9 was obtained by the framework which shows outstanding discrimination of the feature variables used. These were tested centred on the utmost information gain computed, attaining 95.012% (approximately 95%) confidence interval AUC that is 0.97 ± 0.006 . ROC curve and AUC represent the plot of sensitivity ver-sus 1-Specificity as a means of efficiently determining the framework's accuracy with meaningful elucidation. ROC curve plays a fundamental function in assessing the analytical capability of the framework to differentiate the true subject states to get the best cuff of values. This helps in comparing alternative models or investigative tasks that were performed under the same subject. In summary, the ROC curve was used to evaluate and c compare the performance of the APC classifier. We conducted a survey on previous works on Android permissions classifications using general machine learning algorithms, deep learning methods, and the neural networks approach. We focused our comparison on six (6) performance metrics namely: the classifier's accuracy, precision, recall, False Positive Rate (FPR) of the classifier's value, the F-measure values, and the AUC.

Table 6-10 Performance comparison with traditional classifiers

Classifie	ACC		AUC			T_r
	PCA_E	PCA_D	PCA_E	PCA_D	PCA_E	PCA_D
KNB	86.2	89.2	0.92	0.93	12.713	9.355
CSVM	92.5	93.7	0.91	0.92	9.989	5.067
CDT	89.9	92.2	0.90	0.93	4.204	2.0134
ABT	89.9	93.2	0.93	0.94	13.755	11.961
WKNN	90.5	93.7	0.86	0.90	3.045	2.339

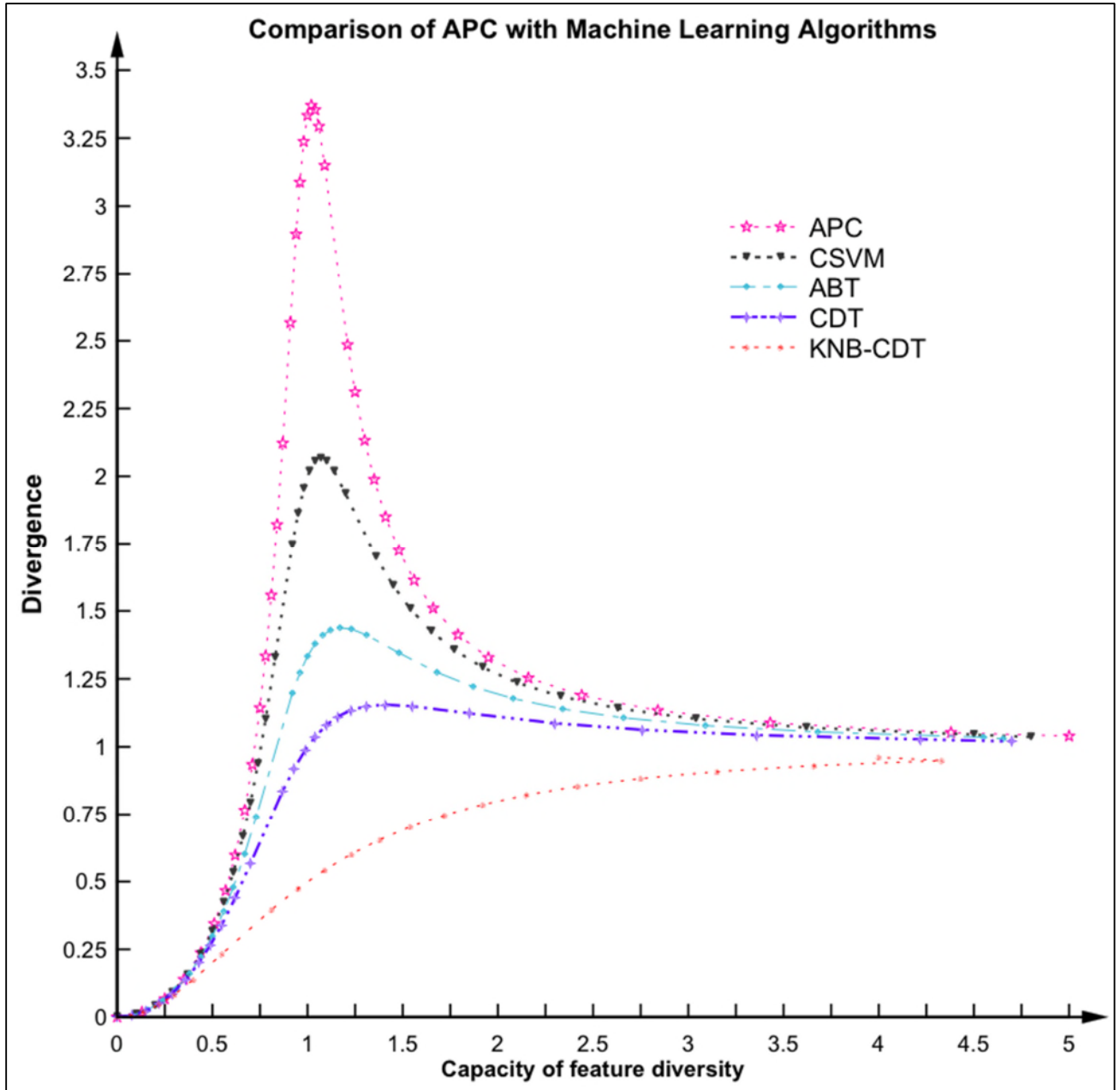
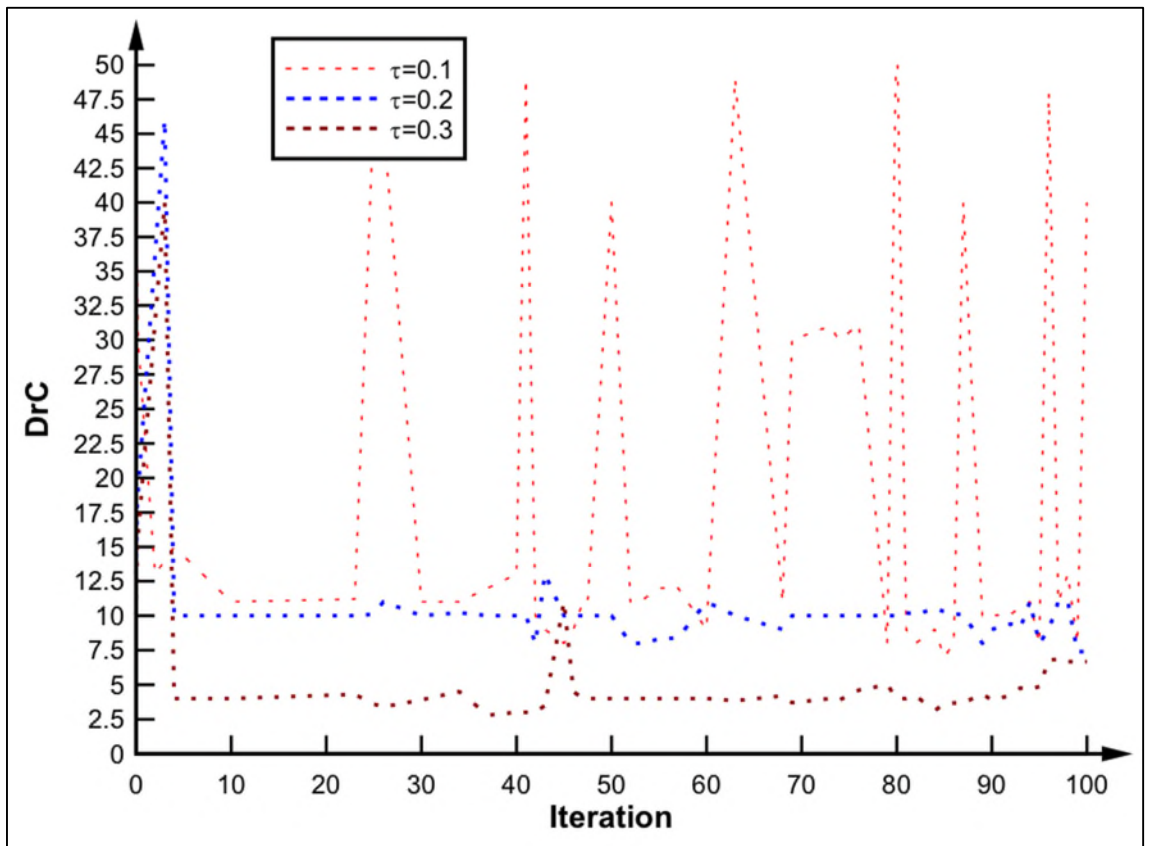


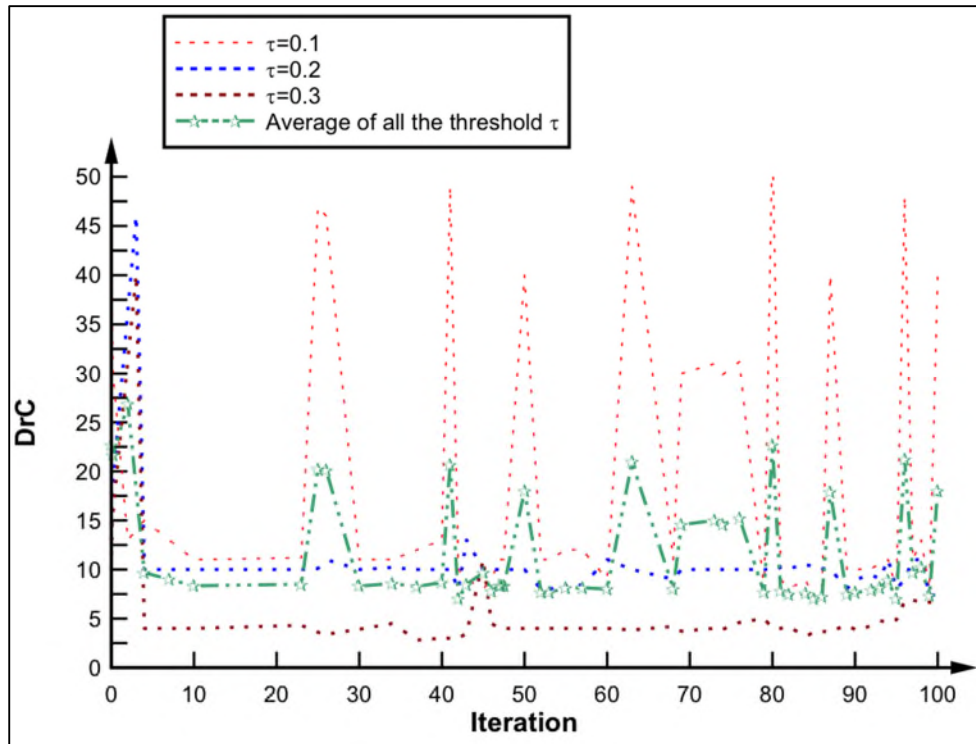
Figure 6-4 Feature diversity of the APC framework compared with existing classifiers

The comparative represented of the existing frameworks. APC: Our framework. The frameworks in the research of [71], [34], and [72] are the previously proposed frameworks. FPR: is the false positive rate for each framework, AUC represents the AUC. F-measure is the F1 Score which measures the test's accuracy of the binary classifiers. As shown in Table III, the machine learning detection approach proposed by Peiravian and Zhu [72] using permission and API calls obtained an accuracy value of 0.18% lower than the framework proposed by Aung and Zaw [34] and 3.84% lower than [71]. Although all the frameworks [71], [34], [72] used a similar approach to feature

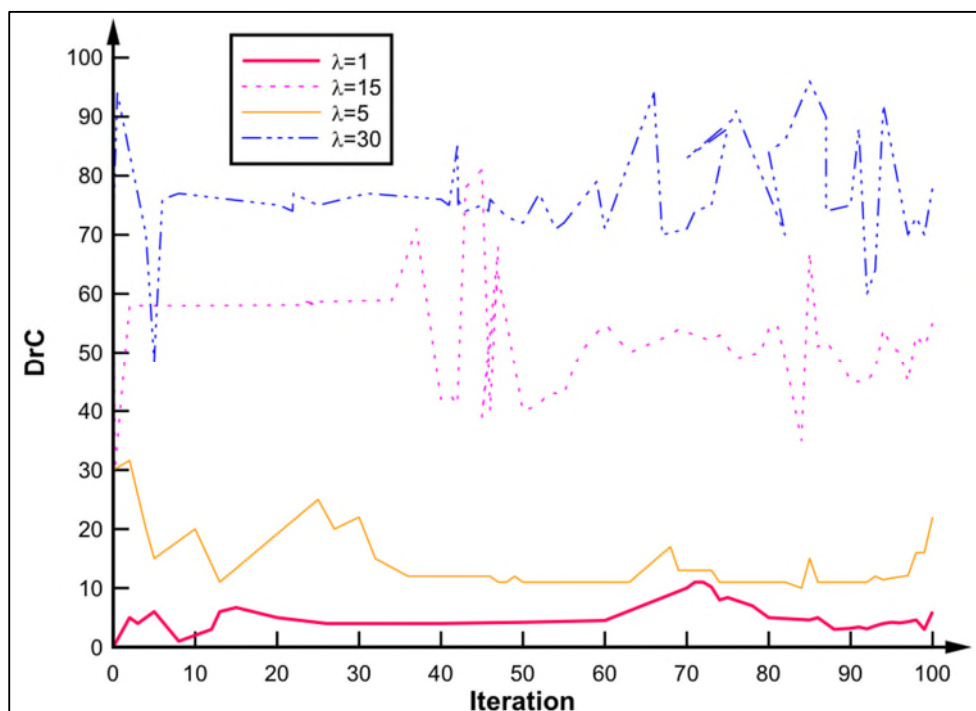
extraction, the method proposed in Xiong et al. [71] focused on contrasting permission patterns and obtained a very low FPR compared to Aung and Zaw and Peiravian and Zhu. Our framework reached an FPR value of 0.20, which is significantly lower than the surveyed frameworks, and 97% classification accuracy. Finally, unlike the previous works, our framework extracts various feature vector sets categories and was not limited to opcode sequence features. By this, the generalization and application of our framework when using diverse benign and malicious attributes become more effective. While the proposed approach of [9] focused on opcode sequence and frequency features, our framework centralized on the feature selection principle of centroid methods.



(A) Threshold parameter



(B) Threshold parameter



(C) Threshold parameter

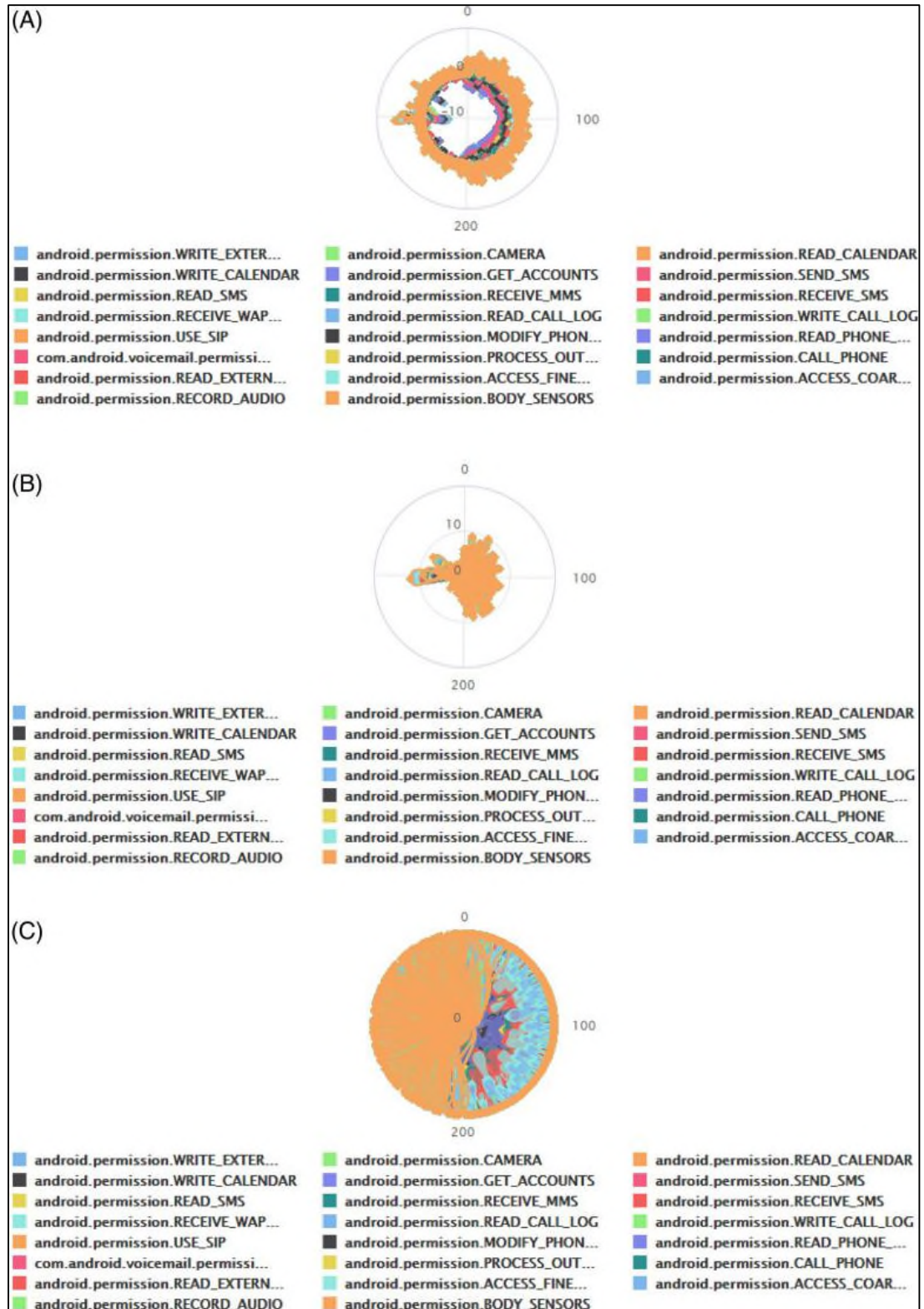
Figure 6-5 Effects of threshold and diversity factor on feature regularisation

6.8.3 Evaluating feature diversity of the APC model

We compared our framework with five traditional machine learning classifiers namely: Kernel Naïve Bayes (KNB), Cubic Support Vector Machine (KSVM), Weighted K Nearest Neighbor (WKNN), Ada Boosted Tree (ABT), and Coarse Decision Tree (CDT) to compare their capacity of feature diversity performance (as shown in Table 10). In each of the classifiers, we applied Principal Components Analysis (PCA). PCA approach enhances feature vector transformation and reduces redundancy in their dimensions. Two different experiments were conducted during PCA application on the classifiers. In the first experiment, the PCA was enabled to help the classifiers keep enough components to explain up to 95% variance. After the training, we observed that irrelevant components were kept, and only variances of the most significant components will be shown.

We observed that higher accuracy was achieved when PCA was disabled than when it was disabled during the classification. In addition, more training time was required for each of the classifiers to finish classification when PCA was enabled. Less training time was taken at disabled PCA. PCA_E : Represents Principal Components Analysis enabled. PCA_D : represents Principal Components Analysis Disabled. T_i represents the training time when PCA was both enabled and disabled. Our framework obtains a divergence value of 3.4672 and a feature diversity capacity of 4.583, respectively (as shown in Figure 6-4). The capacity of feature diversity of APC is higher compared to the survey classifiers in this study.

The effect of threshold and diversity factors on feature regularization (as shown in Figure 6-5) are determined at different threshold parameter values and iterations. The blue line from (b) indicates that the feature diversity increases as the diversity factor increases. Also, when the threshold parameter decreases, the diversity cost increases as in the yellow line of (a) and (c), respectively. During feature diversity experimentation, it was observed that the regularization cost for feature vector diversity increases with a decrease in threshold parameter and the diversity factor respectively.



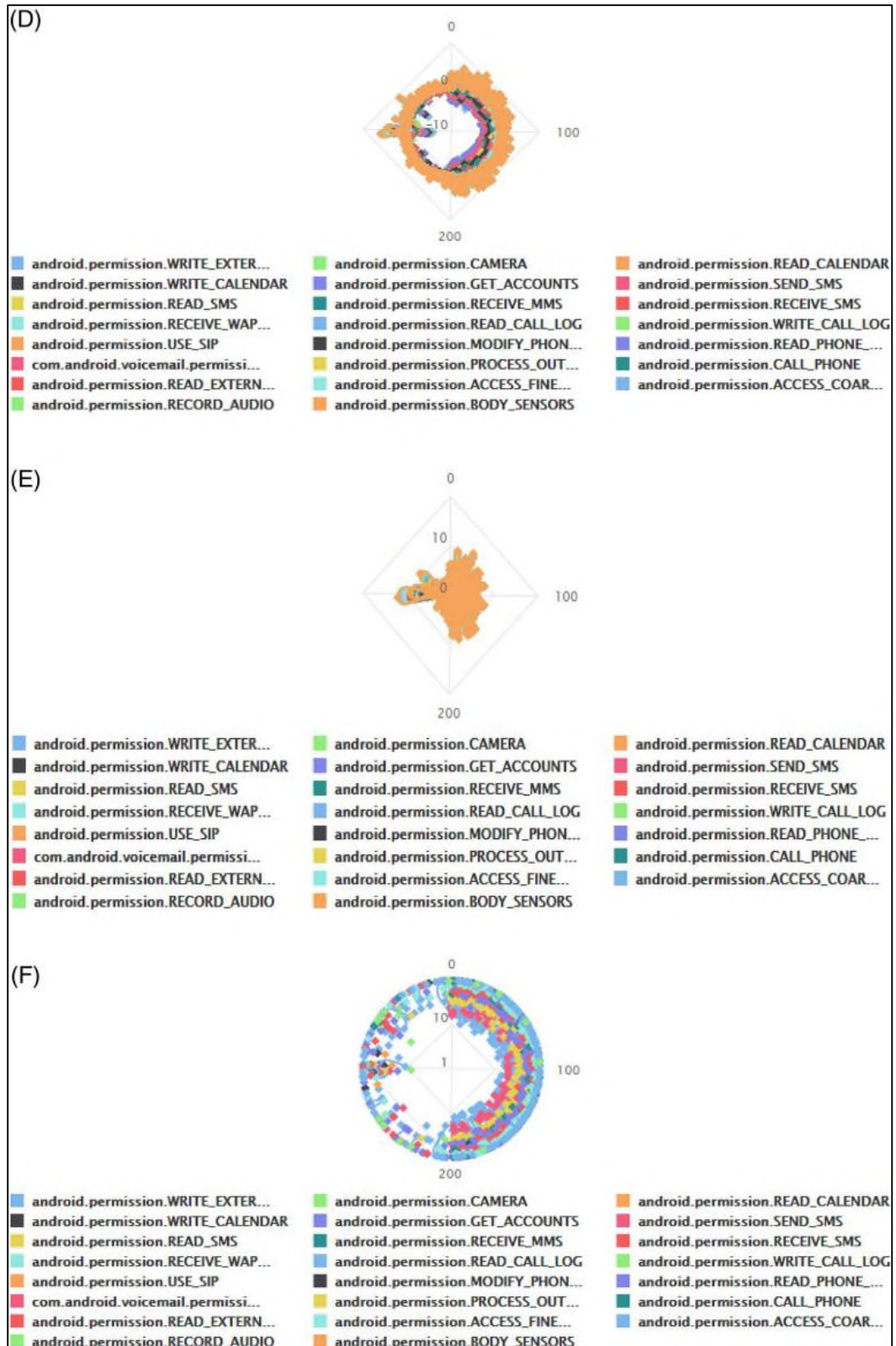


Figure 6-6 Dangerous permissions classification by the framework

6.8.4 Evaluating permission and threat levels

Using the association rule, the result shows that some permissions appeared frequently together. Permissions such as READ_SMS and WRITE_SMS have 97.207% of appearance. They were also frequently requested. We also found that READ_EXTERNAL_STORAGE and WRITE_EXTERNAL_STORAGE_ have a 91.503% chance to appear together. Our framework consistently produces a similar pattern of recall rates. APC classified 24 permissions as having a very high threat level to Android protection based on their frequency of occurrence and request. Permission classification was based on polarised and radared feature vector distribution (as shown in Figure 6-6). The topological data representation and analysis of dangerous features are classified. Figure (6-6a) are the permissions features when the feature set was polarized without stacking, (6-6b) were classified with polarized stacking values, (6-6c) when the training dataset values were stacked to 100 permission features, (6-6d) when the dataset was radared without values been stacked, (e) radared with stacking values, (6-6f) when the radared dataset stacked to 100. When the permission feature vectors were polarized with and without stacking by the framework, the same permission features were classified as having a high threat level to the protection level. The details of the permissions with their respective threat levels (as shown in Figure 6-7).

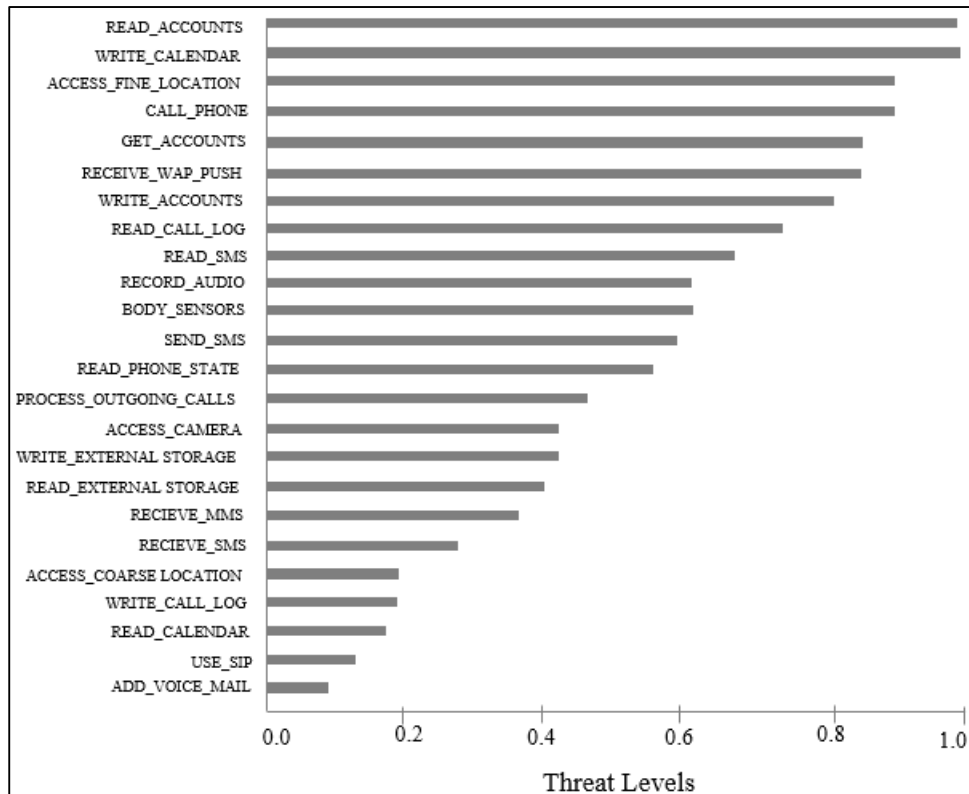


Figure 6-7 Classified permissions with different threat levels by our framework

6.9 Conclusion

Android is the most popular mobile OS platform in recent times. The popularity of Android brought a corresponding increase in the number of attacks targeting the OS. One of the most common and well-known attacks highly experienced by the OS is malware. Malware attacks on Android have caused several security risks ranging from device damage, privacy breach, to financial loss. Different detection and classification tools designed focus chiefly on malware classification and detection no traction to how permissions are used as attack vectors. This article aims to develop a framework for classifying android permission requests to improve the security and protection level of android mobile devices. Our research shows that the accuracy of permission requests by malicious applications can be enhanced by using an effective classification framework. Classification of android permissions with high threat levels can be improved without having much impact on the classification accuracy by using a regularization approach to increase feature diversity selection procedure. We proposed a new android permission classification framework. The best performance, obtained by combining regularization and feature diversity scheme, is a 97% classification accuracy of the model.

In general, a total of four distinct features were extracted from different APK file segments. We used feature vectors such as API calls, command signatures, permission requests to train the network. To increase the classifiers' performance run time, 113 permissions were considered with 97% classification accuracy. Other learning algorithms tested achieved 92.4% accuracy on average when PCA was disabled and 89.8% when PCA was disabled, respectively. The results of the experiments presented in this work show a higher AUC achievement of 95% compared with the state-of-the-art permission request classification and other machine learning classifiers.

We suggested a regularization function for feature diversity to reduce filtering and FRC. This helps to increase feature diversity selection procedure and generalization of the framework improvement. Our framework has a higher capacity for feature diversity when compared with traditional machine learning algorithms. Our framework classified 24 permissions as having a higher threat level to Android protection level than the rest of the permissions. Notwithstanding the positive performance result obtained, additional research must be done to improve the accuracy and synthesis of the relationship between threat and protection level of the permission request. This forms our future work and the direction for future research.

ACKNOWLEDGMENTS

This research is supported by the Petroleum Technology Development Fund (PTDF) by a grant PTDF/ED/PHD/AMA/1245/17.

REFERENCES

- [1] M. Ashawa, N. Ogwuche, "Forensic Data Extraction and Analysis of Left Artifacts on emulated Android Phones : A Case Study of Instant Messaging Applications," *Circulation in computer science*, vol.2, no. 11, pp. 8-16, 2017, doi: 10.22632/ccs-2017-252-67.
- [2] M. Ashawa, A. Mansour, M. A. Haider, "Extracting data from Android's instant messaging and social media apps", *Quarterly Magazine for Digital Forensics Practitioners*, vol. 33, no. 1, 2017.
- [3] IDC, "Smartphone Market Share," The International Data Corporation (IDC) Worldwide Quarterly Mobile Phone. Accessed on: Jan. 18, 2020. [Online]. Available: <https://www.idc.com/promo/smartphone-market-share/os>.
- [4] T. Kim, B. Kang, M. Rho, S. Sezer and E. G. Im, "A Multimodal Deep Learning Method for Android Malware Detection Using Various Features," in *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 3, pp. 773-788, 2019, doi: 10.1109/TIFS.2018.2866319.
- [5] L. Hoang, T. Nguyen, T. Cam, and V. Pham, "Enhancing the accuracy of static analysis for detecting sensitive data leakage in Android by using dynamic analysis," *Cluster Comput.*, vol. 22, no.1, pp. 1079–1085, 2019, doi: 10.1007/s10586-017-1364-8.
- [6] Z. Meng, Y. Xiong, W. Huang, L. Qin, X. Jin, and H. Yan, "Neurocomputing AppScalpel : Combining static analysis and outlier detection to identify and prune undesirable usage of sensitive data in Android applications," *Neurocomputing*, vol. 341, pp. 10–25, 2019, doi: 10.1016/j.neucom.2019.01.105.
- [7] A. T. Kabakus, "What Static Analysis Can Utmost Offer for Android Malware Detection," *Journal of Information Technology and Control*, vol. 48, no. 2, pp. 235–249, 2019, doi: 10.5755/j01.itc.48.2.21457.
- [8] R. Vinayakumar, K. P. Soman, P. Poornachandran, S.S Sachin Kumar, "Detecting Android malware using long short-term memory (LSTM)". *Journal of Intelligent and Fuzzy System*, vol. 34, no. 3, pp. 1277-1288.
- [9] K. Bakour, H. M. Ünver and R. Ghanem, "The Android Malware Static Analysis: Techniques, Limitations, and Open Challenges," 2018 3rd International Conference on Computer Science and Engineering (UBMK), 2018, pp. 586-593, doi: 10.1109/UBMK.2018.8566573.
- [10] A. Firdaus, N. B. Anuar, A. Karim, M. Faizal, and A. Razak, "Discovering optimal features using static analysis and a genetic search based method for Android malware detection," *Frontiers Information Technology Electronic Engineering*, vol. 19, no. 6, pp. 712–736, 2018, doi: <https://doi.org/10.1631/FITEE.1601491>.
- [11] J. Li, L. Sun, Q. Yan, Z. Li, W. Srisa-an and H. Ye, "Significant Permission Identification for Machine-Learning-Based Android Malware Detection," in *IEEE Transactions on Industrial Informatics*, vol. 14, no. 7, pp. 3216-3225, July 2018, doi: 10.1109/TII.2017.2789219.
- [12] H. Zhu, Z. You, Z. Zhu, W. Shi, X. Chen, and L. Cheng, "Neurocomputing DroidDet : Effective and robust detection of android malware using static analysis along with rotation forest model," *Neurocomputing*, vol. 272, pp. 638–646, 2018, doi: 10.1016/j.neucom.2017.07.030.
- [13] W. Y. Lee, J. Saxe, R. Harang, "SeqDroid: Obfuscated android malware detection using stacked convolutional and recurrent neural networks". *Deep learning applications for cyber security*. Cham: Springer, pp. 197-210, 2019.
- [14] X. Xiao, S. Zhang, and F. Mercaldo, "Android malware detection based on system call sequences and LSTM," *Multimed Tools Appl*, vol. 78, pp. 3979–3999, 2019, doi: 10.1007/s11042-017-5104-0.
- [15] S. Das, A. E. Dubrovsky, I. Korsunsky, A. Dhablania, J.E.G. Muender, "Just in time memory analysis for malware detection," U.S. Patent Application 15/890,192. SonicWALL Inc, 2019.

- [16] J. Lalande and P. Berthomé, "Teaching Android Mobile Security," Proceedings of the 50th ACM Technical Symposium on Computer Science Education, 27 March 2019, Minneapolis, MN, USA, pp. 232–238, 2019.
- [17] N. Nissim, O. Lahav, A. Cohen, Y. Elovici, and L. Rokach, "Computers & Security Volatile memory analysis using the MinHash method for efficient and secured detection of malware in private cloud," *Comput. Secur.*, vol. 87, p. 101590, 2019, doi: 10.1016/j.cose.2019.101590.
- [18] Z. Grimmer, J. Staggs, S. Sheno, "Retrofitting Mobile Devices for Capturing Memory-Resident Malware Based on System Side-Effects," *IFIP Int Conf Dig Forens.* 589. Cham, Switzerland: Springer Nature Switzerland, pp. 59-72, 2019, doi: https://doi.org/10.1007/978-3-030-28752-8_4.
- [19] C. Zhao, C. Wang, and W. Zheng, "Android Malware Detection Based on Sensitive Permissions and APIs," in *Security and Privacy in New Computing Environments*, vol. 284, pp. 105–113, 2019, doi: https://doi.org/10.1007/978-3-030-21373-2_10.
- [20] W. -C. Kuo, T. -P. Liu and C. -C. Wang, "Study on Android Hybrid Malware Detection Based on Machine Learning," 2019 IEEE 4th International Conference on Computer and Communication Systems (ICCCS), 23-25 Feb. 2019, Singapore, pp. 31-35, doi: 10.1109/CCOMS.2019.8821665.
- [21] W. Wang, M. Zhao, and J. Wang, "Effective android malware detection with a hybrid model based on deep autoencoder and convolutional neural network," *J. Ambient Intell. Humaniz. Comput.*, vol. 10, no. 8, pp. 3035–3043, 2019, doi: 10.1007/s12652-018-0803-6.
- [22] S. Kothari, "Real Time Analysis of Android Applications by Calculating Risk Factor to Identify Botnet Attack," in *ICCCE 2019, Lecture Notes in Electrical Engineering*, vol. 570. Springer, Singapore, 2019, doi: https://doi.org/10.1007/978-981-13-8715-9_7.
- [23] J. Lopes, C. Serrão, L. Nunes, A. Almeida and J. Oliveira, "Overview of machine learning methods for Android malware identification," 2019 7th International Symposium on Digital Forensics and Security (ISDFS), 10-12 June 2019, Barcelos, Portugal, pp. 1-6, 2019, doi: 10.1109/ISDFS.2019.8757523.
- [24] F. C. Colon Osorio, H. Qiu and A. Arrott, "Segmented sandboxing - A novel approach to Malware polymorphism detection," 2015 10th International Conference on Malicious and Unwanted Software (MALWARE), 20-22 Oct. 2015, Fajardo, PR, USA, pp. 59-68, doi: 10.1109/MALWARE.2015.7413685.
- [25] M. Ashawa and S. Morris, "Analysis of Android Malware Detection Techniques: A Systematic Review," *Int. J. Cyber-Security Digit. Forensics*, vol. 8, no. 3, pp. 177–187, 2019, doi: 10.17781/p002605.
- [26] Y. L. Arnatovich, L. Wang, N. M. Ngo and C. Soh, "A Comparison of Android Reverse Engineering Tools via Program Behaviors Validation Based on Intermediate Languages Transformation," in *IEEE Access*, vol. 6, pp. 12382-12394, 2018, doi: 10.1109/ACCESS.2018.2808340.
- [27] K. A. Talha, D. I. Alper, and C. Aydin, "APK Auditor: Permission-based Android malware detection system," *Digit. Investig.*, vol. 13, pp. 1–14, 2015, doi: <https://doi.org/10.1016/j.diin.2015.01.001>.
- [28] Androzoo. Accessed on: Jan. 2019. [Online]. Available: <https://androzoo.uni.lu/>
- [29] Android Malware Genome Project. Accessed on: Jan. 2019. [Online]. Available: <http://www.malgenomeproject.org/>.
- [30] VirusShare. Accessed on: Jan. 2019. [Online]. Available: <https://virusshare.com>
- [31] Contagio Mobile. Accessed on: Jan. 2019. [Online]. Available: <http://contagiomobile.deependresearch.org/index.html>.

- [32] M. Ashawa, S. Morris, "Host-based detection and analysis of Android malware: implication for privilege exploitation". *International Journal for Information Security Research*, vol. 9, no. 2, pp. 871-880, 2019.
- [33] C.-Y. Huang, Y.-T. Tsai, and C.-H. Hsu, "Performance Evaluation on Permission-Based Detection for Android Malware," in *Advances in Intelligent Systems and Applications - Volume 2*, 2013, pp. 111–120.
- [34] Z. Aung And W. Zaw, "Permission-Based Android Malware Detection," *International Journal Of Scientific & Technology Research*, vol. 2, no. 3, pp. 228-234, 2013.
- [35] D. Arp, M. Spreitzenbarth, H. Malte, H. Gascon, and K. Rieck, "D REBIN: Effective and Explainable Detection of Android Malware in Your Pocket." *Proc Netw Distrib Syst Secur Symp (NDSS)*, vol. 14, pp. 23-26, 2014.
- [36] A. Narayanan, L. Yang, L. Chen and L. Jinliang, "Adaptive and scalable Android malware detection through online learning," *International Joint Conference on Neural Networks (IJCNN)*, 24-29 July 2016, Vancouver, BC, Canada, pp. 2484-2491, doi: 10.1109/IJCNN.2016.7727508.
- [37] Enck W, Gilbert P, Chun BG, et al. TaintDroid: An information flow tracking system for real time privacy monitoring on smartphones. *ACM Trans Comput Syst.*, vol. 57, New York, NY, USA, pp. 1-5, 2014.
- [38] L. K. Yan, "DroidScope : Seamlessly Reconstructing the OS and Dalvik Semantic Views for Dynamic Android Malware Analysis." In *21st {USENIX} Security Symposium ({USENIX} Security*, vol. 12, pp. 569-584.
- [39] A. S. F. Peters, F. Lamour, C. Scheel, S. Ahmet, and Ç. Sahin, "Monitoring Smartphones for Anomaly Detection," *Mobile Netw Appl*, vol. 14, pp. 92–106, 2009, doi: 10.1007/s11036-008-0113-x.
- [40] N. McLaughlin et al., "Deep android malware detection," *CODASPY 2017 - Proc. 7th ACM Conf. Data Appl. Secur. Priv.*, pp. 301–308, 2017, doi: 10.1145/3029806.3029823.
- [41] J. Saxe and K. Berlin, "Deep neural network based malware detection using two dimensional binary program features," *2015 10th International Conference on Malicious and Unwanted Software (MALWARE)*, 20-22 Oct. 2015, Fajardo, PR, USA pp. 11-20, doi: 10.1109/MALWARE.2015.7413680.
- [42] O. E. David and N. S. Netanyahu, "DeepSign: Deep learning for automatic malware signature generation and classification," *2015 International Joint Conference on Neural Networks (IJCNN)*, 12-17 July 2015, Killarney, Ireland, pp. 1-8, doi: 10.1109/IJCNN.2015.7280815.
- [43] T. Kim, B. Kang, M. Rho, S. Sezer and E. G. Im, "A Multimodal Deep Learning Method for Android Malware Detection Using Various Features," in *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 3, pp. 773-788, March 2019, doi: 10.1109/TIFS.2018.2866319.
- [44] B. Kang, S. Y. Yerima, K. Mclaughlin and S. Sezer, "N-opcode analysis for android malware classification and categorization," *2016 International Conference On Cyber Security And Protection Of Digital Services (Cyber Security)*, 2016, pp. 1-7, doi: 10.1109/CyberSecPODS.2016.7502343.
- [45] W. Wang, X. Wang, D. Feng, J. Liu, Z. Han and X. Zhang, "Exploring Permission-Induced Risk in Android Applications for Malicious Application Detection," in *IEEE Transactions on Information Forensics and Security*, vol. 9, no. 11, pp. 1869-1882, Nov. 2014, doi: 10.1109/TIFS.2014.2353996.
- [46] S. Yerima, Android malware dataset for machine learning". Accessed on: Jan. 2019. [Online]. Available: https://figshare.com/articles/Android_malware_dataset_for_machine_learning_2/5854653.

- [47] C. Hasegawa and H. Iyatomi, "One-dimensional convolutional neural networks for Android malware detection," 2018 IEEE 14th International Colloquium on Signal Processing & Its Applications (CSPA), 9-10 March 2018, Penang, Malaysia pp. 99-102, doi: 10.1109/CSPA.2018.8368693.
- [48] J. Hao, G. Zhang, Y. Zheng, W. Hu and K. Yang, "Solution for Selective Harmonic Elimination in Asymmetric Multilevel Inverter Based on Stochastic Configuration Network and Levenberg-Marquardt Algorithm," 2019 IEEE Applied Power Electronics Conference and Exposition (APEC), 17-21 March 2019, Anaheim, CA, USA, pp. 2855-2858, doi: 10.1109/APEC.2019.8722057.
- [49] B. Kim, K. Lim, S. Cho and M. Park, "RomaDroid: A Robust and Efficient Technique for Detecting Android App Clones Using a Tree Structure and Components of Each App's Manifest File," in IEEE Access, vol. 7, pp. 72182-72196, 2019, doi: 10.1109/ACCESS.2019.2920314.
- [50] B. Vasu and N. Pari, "Combining Multimodal DNN and SigPid technique for detecting Malicious Android Apps," 2019 11th International Conference on Advanced Computing (ICoAC), 18-20 Dec. 2019, Chennai, India, pp. 289-294, doi: 10.1109/ICoAC48765.2019.247134.
- [51] W. Pei, J. Li, H. Li, H. Gao, P. Wang, "ASCAA: API-level security certification of android applications". IET Softw. vol. 11, no. 2, 2017.
- [52] A. K. Jha, S. Lee and W. J. Lee, "Developer Mistakes in Writing Android Manifests: An Empirical Study of Configuration Errors," 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR), 2017, pp. 25-36, doi: 10.1109/MSR.2017.41.
- [53] L. Sun, Z. Li, Q. Yan, W. Srisa-an and Y. Pan, "SigPID: significant permission identification for android malware detection," 2016 11th International Conference on Malicious and Unwanted Software (MALWARE), 2016, pp. 1-8, doi: 10.1109/MALWARE.2016.7888730.
- [54] F. S. U. Near-infrared, W. Kim, J. M. Song, and K. R. Park, "Multimodal Biometric Recognition Based on Convolutional Neural Network by the Fusion of (NIR) Camera Sensor," vol. 18, no. 7, 2018, doi: 10.3390/s18072296.
- [55] F. Firdausillah et al., "Implementation of Neural Network Backpropagation Using Audio Feature Extraction For Classification Of Gamelan Notes," 2018 International Seminar on Application for Technology of Information and Communication, 2018, pp. 570-574, doi: 10.1109/ISEMANTIC.2018.8549805.
- [56] W. C. Lin and Y. H. Lu, "Feature selection in single and ensemble learning - based bankruptcy prediction models," Expert systems, vol. 36, no. 1, 2018, pp. 1-8, 2019, doi: 10.1111/exsy.12335.
- [57] J. Li, L. Sun, Q. Yan, Z. Li, W. Srisa-an and H. Ye, "Significant Permission Identification for Machine-Learning-Based Android Malware Detection," in IEEE Transactions on Industrial Informatics, vol. 14, no. 7, pp. 3216-3225, July 2018, doi: 10.1109/TII.2017.2789219.
- [58] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks," in Computer Vision -- ECCV 2016, 2016, pp. 525-542.
- [59] J. Gu et al., "Recent advances in convolutional neural networks," Pattern Recognit., vol. 77, pp. 354-377, 2018, doi: 10.1016/j.patcog.2017.10.013.
- [60] N. A. Khafaf and A. El-Hag, "Bayesian regularization of neural network to predict leakage current in a salt fog environment," in IEEE Transactions on Dielectrics and Electrical Insulation, vol. 25, no. 2, pp. 686-693, April 2018, doi: 10.1109/TDEI.2017.006936.
- [61] D. T. Bartilson, J. Jang, and A. W. Smyth, "Finite element model updating using objective-consistent sensitivity-based parameter clustering and Bayesian regularization," Mech. Syst. Signal Process., vol. 114, pp. 328-345, 2019, doi: 10.1016/j.ymssp.2018.05.024.

- [62] N. Shiratori, "Modeling Dropout Behavior Patterns Using Bayesian Networks in Small-Scale Private University," 2017 6th IIAI International Congress on Advanced Applied Informatics (IIAI-AAI), 2017, pp. 170-173, doi: 10.1109/IIAI-AAI.2017.178.
- [63] M. Kayri, "Predictive Abilities of Bayesian Regularization and Levenberg – Marquardt Algorithms in Artificial Neural Networks : A Comparative Empirical Study on Social Data," 2016, doi: 10.3390/mca21020020.
- [64] B. O. Ayinde, T. Inanc and J. M. Zurada, "Regularizing Deep Neural Networks by Enhancing Diversity in Feature Extraction," in IEEE Transactions on Neural Networks and Learning Systems, vol. 30, no. 9, pp. 2650-2661, Sept. 2019, doi: 10.1109/TNNLS.2018.2885972.
- [65] N. Ketkar, "Introduction to Keras," in Deep Learning with Python: A Hands-on Introduction, Berkeley, CA: Apress, 2017, pp. 97–111.
- [66] F. Ertam and G. Aydin, "Data classification with deep learning using Tensorflow," 2017 International Conference on Computer Science and Engineering (UBMK), 5-8 Oct. 2017, Antalya, Turkey, pp. 755-758, doi: 10.1109/UBMK.2017.8093521.
- [67] A. Géron, "Hands-on machine learning with Scikit-Learn and TensorFlow". Boston, MA: O'Reilly Media, 2019.
- [68] M. Hall et al., "The WEKA Data Mining Software : An Update," ACM SIGKDD explorations newsletter, vol. 11, no. 1, pp.10-18, 2009, doi: <https://doi.org/10.1145/1656274.1656278>.
- [69] T. Fawcett, "An introduction to ROC analysis," vol. 27, no. 8, pp. 861–874, 2006, doi: 10.1016/j.patrec.2005.10.010.
- [70] T. Fawcett, "ROC Graphs : Notes and Practical Considerations for Researchers," HP Inven, vol. 6, no. 1, pp. 1–38, 2004.
- [71] P. Xiong, X. Wang, W. Niu, T. Zhu and G. Li, "Android malware detection with contrasting permission patterns," in China Communications, vol. 11, no. 8, pp. 1-14, Aug. 2014, doi: 10.1109/CC.2014.6911083.
- [72] N. Peiravian and X. Zhu, "Machine Learning for Android Malware Detection Using Permission and API Calls," 2013 IEEE 25th International Conference on Tools with Artificial Intelligence, 4-6 Nov. 2013, Herndon, VA, USA, pp. 300-305, doi: 10.1109/ICTAI.2013.53.

CHAPTER 7: MODELLING CORRELATION BETWEEN ANDROID PERMISSION VARIABLES BASED ON THREAT AND PROTECTION LEVEL USING EXPLORATORY FACTOR PLANE ANALYSIS

The content of this chapter is published in the International Journal of Cybersecurity and Privacy, MDPI Publisher.

Citation: M. Ashawa and S. Morris, “Modelling Correlation between Android Permissions Based on Threat and Protection Level Using Exploratory Factor Plane Analysis,” Journal of Cybersecurity and Privacy, vol. 1, no. 4, pp. 704–743, 2021.

As discussed in the previous chapter, dangerous permissions are often requested by malicious applications during the installation process. Oftentimes, it is very difficult to distinguish if the permission requested has a high-risk level if granted. Thus, examining the correlation between permissions to identify their threat and protection level in an application is significant. The understanding of high threat permission will assist in enforcing security parameters at the permission security system of the Android platform and other mobile OS. These findings are significant for offering insights to mobile users to aid in making low-risk decisions during application installations and when granting access to applications that have high threat levels especially, especially when similar permission names with the same protection level are requested.

This chapter is important because understanding the relationship between permission variables is significant in exploring factors that affect Android application libraries especially when private interface calls are invoked. During the Android runtime process, remote procedure handling API calls are initiated for both the protection level and permission flags. While services such as incident approver are instantiated by the permission flags, protection level flags are instantiated by different permissions: normal, dangerous, signature, and SignatureOrSystem. The model identified the correlation between permission variables with flags that indicate the presence of malicious based on their threat levels. This is to facilitate analysis of sophisticated malware which compromise users’ confidentiality and general privacy using dangerous permissions and intents, among others.

Abstract

The evolution of mobile technology has increased correspondingly with the number of attacks on mobile devices. Malware attack on mobile devices is one of the top security challenges the mobile community faces daily. While malware classification and detection tools are being developed to fight malware infection, hackers keep deploying different infection strategies, including permissions usage. Among mobile platforms, Android is the most targeted by malware because of its open OS and popularity. Permissions is one of the major security techniques used by Android and other mobile platforms to control device resources and enhance access control. In this study, we used the t-Distribution stochastic neighbor embedding (t-SNE) and Self-Organizing Map techniques to produce a visualization method using exploratory factor plane analysis to visualize permissions correlation in Android applications. Two categories of datasets were used for this study: the benign and malicious datasets. Dataset was obtained from Contagio, VirusShare, VirusTotal, and Androzoo repositories. A total of 12,267 malicious and 10,837 benign applications with different categories were used. We demonstrate that our method can identify the correlation between permissions and classify Android applications based on their protection and threat level. Our results show that every permission has a threat level. This signifies those permissions with the same protection level have the same threat level.

Keywords: Cybersecurity; mobile malware; factor analysis; dangerous permission variables; protection level; Bayesian correlation; threat level

7.0 Introduction

Among the smartphone platforms, Android is the most popular. A previous study [1] showed that Android owns 74.5% of the mobile device marketplace. The popularity of Android has caused an increase in third-party application development to respond to the shift. Factors such as open-source [2-3], ease of use [4-5], and low cost [6-7] contribute significantly to its popularity and fast spread. The open-source nature of the Android OS architecture enables the unrestricted distribution of third-part applications on different distribution markets. Consequently, a wide-ranging API is provided to third-party developers. The large-scale permits Android applications to have access to the device resources and users' privacy when certain permissions are accepted by the user.

While some of the permissions are normal, Google classified certain permission variables as dangerous. Permissions variables that are classified as dangerous would likely have a higher threat level than normal permissions. To ensure that users' data are protected, Android imposes applications to request and obtain permissions before accessing sensitive data from the device. During application installation, if the user denies granting access, installation is aborted. Any permission request to access device resources and users' private information can potentially be dangerous due to the advent of millions of malicious applications using different infection and evasion techniques. Anytime an application requests access to an Android device, the request forces the user to decide whether to accept the decision or decline the permission. While some permissions have malicious intent, most mobile users do not think of the implication of granting the permissions. Still, they are more concerned with getting the application downloaded and installed, as outlined in the research of [8]. The results obtained in the study of [9] affirm that the majority of mobile users are careless about accepting permissions than the resultant havoc such permission could cause.

While few users pay attention to the kind of permission access request by applications, malware writers deploy different ways to drip them by setting some malicious applications with default handlers to avoid multiple permission requests during runtime [10]. It then implies that once certain permission is accepted to access the device resources, others with similar features are likely to be installed in the background without presenting a further request for access [11]. The study of [12] identified that in a random sample of a given population, features of some variables have control or influence the characteristics of one or more variables at given levels of independent measurement. This signifies that acceptance of specific permissions influence the installations of the others with a certain level of correlation. It then means there is a correlation that exists between permission variables. The open issue then is how to control the behaviour of permissions that malicious applications request. To achieve this, understanding the relationship between permission variables is key. Therefore, since the security of Android depends on the performance of the permission mechanism of the OS platform, it is pertinent to determine how correlated permissions are. Understanding how permissions are related will provide insights into how Android applications handle sensitive and high-risk data. Additionally, this will help in human decision making when installing applications whose source is not legitimate. The key contributions of our research are the following:

- 1) We present a model for visualizing the correlation between Android permissions using the t-Distribution stochastic neighbor embedding (t-SNE) and Self-Organizing Map (SOM) techniques.
- 2) We demonstrate results that show the relationship between a threat and protection level in Android permissions using exploratory factor plane analysis. The results show that every permission, whether normal or dangerous, has a threat level.
- 3) We identify that Android permissions with the same protection level have the same threat level. However, the threat level in the individual applications differ.
- 4) To examine Android permissions commonly requested and disseminated to classify Android applications as malicious or benign. Our results demonstrate that the proposed model can determine families of malware based on the similarities by understanding their clusters. This demonstrates that Android permissions in the same cluster have similar attributes.
- 5) We build on the existing work to expand Android permission request state-of-the-art by providing a comprehensive study on the current state of permissions systems.

The remainder of this paper is structured as follows. In Section 7.1, background of the study was provided. Section 7.2 provided the methodology for the research while the results and the discussion were provided in sections 7.4 and 7.5 respectively.

7.1. Background

This section provides background to the Android permission architecture and other components. It also provides detailed background on the protection levels, permission flags, and other essential Android security components such as the API calls and message intents.

7.1.1 Android permission architecture and other components

The security architecture of Android revolved around its permission system. Based on the security architectural design of the Android platform, no application is allowed to install and function in any form if the Android OS, other applications, or users' data are affected adversely.

The security architecture that regulates and enforces this is the permission system. However, due to regular updates in Android versions and code names, the permission system keeps evolving. Additionally, the range of the Android API levels increases with the increase in the number of permissions (Table 7-1). Each Android version released is incorporated with higher API levels and more permission features. For instance, the first Android code name (Base) released has an API level of 2 with total permissions of 73 [13-14]. However, as new code names emerged, there is an increase in the API levels and the permission features. The increase in the total number of permissions is influenced by the protection level offered by the device. The protection level is one of the permission groups defined by Android [15].

Table 7-1 Android code name release versions show the increase in the API level and number of permissions.

Code Name	Platform Version	API Level	Number of Permissions
Android 11 beta	11	30	167
Q	10	29	158
PI	9	28	148
Oreo	8.0–8.1	26–27	144
Nougat	7.0–7.1	24–25	135
Marshmallow	6	23	131
Lollipop	5.0–5.1	21–22	120
KitKat Watch	4.4 W	20	113
KitKat	4.4	19	112
Jelly Bean	4.1–4.3.1	16–18	104
Ice Cream Sandwich	4.0.1–4.0.4	14–15	98
Honeycomb	3.0–3.2	11–13	95
Gingerbread	2.0–2.3.5	9–10	94
Froyo	2.2.	8	87
Éclair	2.0–2.1	5–7	86
Donut	1.6	4	85
Cupcake	1.5	3	81
Base	1–1.1	1–2	73

7.1.2 Protection levels and permission flags

The levels mean the intention of using specific permission and the resultant effects or consequences that come with using such permission. The protection level in the Android system defines the potential threat hidden in the permission. It specifies the possible measure the system should adapt when deciding whether to grant access or deny the permission requested by an application. Android permission system supports four basic types of protection levels with different permission flags (as shown in Figure 7-1). These include normal protection level, dangerous protection level, signature protection level, and signatureOrSystem protection level.

7.1.2.1 Normal protection level

Systems usually grant access to normal protection levels without needing compliance from the user. Normal permissions are sometimes misused by malware to bypass mobile device security to access device resources and other private data. Malware authors may define some functionalities as access control mechanisms to communicate with pre-defined applications to access the resources. For instance, when these permissions are excessively requested by the over-privileged applications, this poses security issues by increasing the threat level of a device.

7.1.2.2 Signature protection level

This protection level compares the certificate of declaring and requesting application to determine if it can be signed by the same certificate as that of the device manufacturer. Permission is granted only if the certification condition is satisfied; that is to say, signature permissions can only be granted if the same certification signs both parties. Signature permissions are only granted if the requesting application is signed by the same developer that defined the permission.

7.1.2.3 Dangerous protection level

Dangerous protection level grants access to the device's resources, private information of the user, device operating system, and the functioning of other applications installed on the device. Dangerous permissions are operational when users grant explicit access to them. On most occasions, dangerous permissions are presented to the user when installing third-party applications.

7.1.2.4 Signature or System protection level

SignatureOrSystem protection level is granted to only applications that are signed with the same system Android system image certificate. SignatureOrSystem permissions are required where applications built by two or more vendors need specific features of the system image to be shared explicitly.

While the pre-installed applications can use permissions in all the protection level categories, only normal and dangerous permissions can be used by third-party applications. Any time permissions from the signature and SignatureOrSystem protection level are requested by third-party applications, such permissions are rejected by the OS platform. Malicious applications can leverage different permission flag constants such as `pre23` [16-17] of the system protection level to automatically grant permissions to malicious applications that target API levels. Applications with permission flags granted automatically reside in the Original Equipment Manufacturer (OEM) partition of the Android. Due to the connection between the OEM and the MaskBase [18] in the protection level, privileged flags such as `vendorPrivileged` enables both malicious and benign applications to access any permission, especially those associated with the system image.

7.1.3 Intent message

Malek defined an Intent message as “a trigger event for an activity or service to be performed along with the required data which supports that requested action” [19]. Intent messages, when exchanged, can be used by malicious applications to escalate privileges. The inter-component communication mechanizing [20] of Android relies chiefly on Intent messages. Intents are generally used by Android to aid data delivery via asynchronous messages. An Intent object is significant in Android permission because it holds data that the Android system uses to define recipient components needed to operate. This is significant because it enhances and eases service implementation in passing data and making it available to applications. This allows the user to initiate an action in a different application by calling a simple operation that the user would like to perform. Examples of such activities include but are not limited to taking pictures or viewing maps [21]. Intent plays a significant role as a communication mechanism in intra-application and inter-application message exchange.

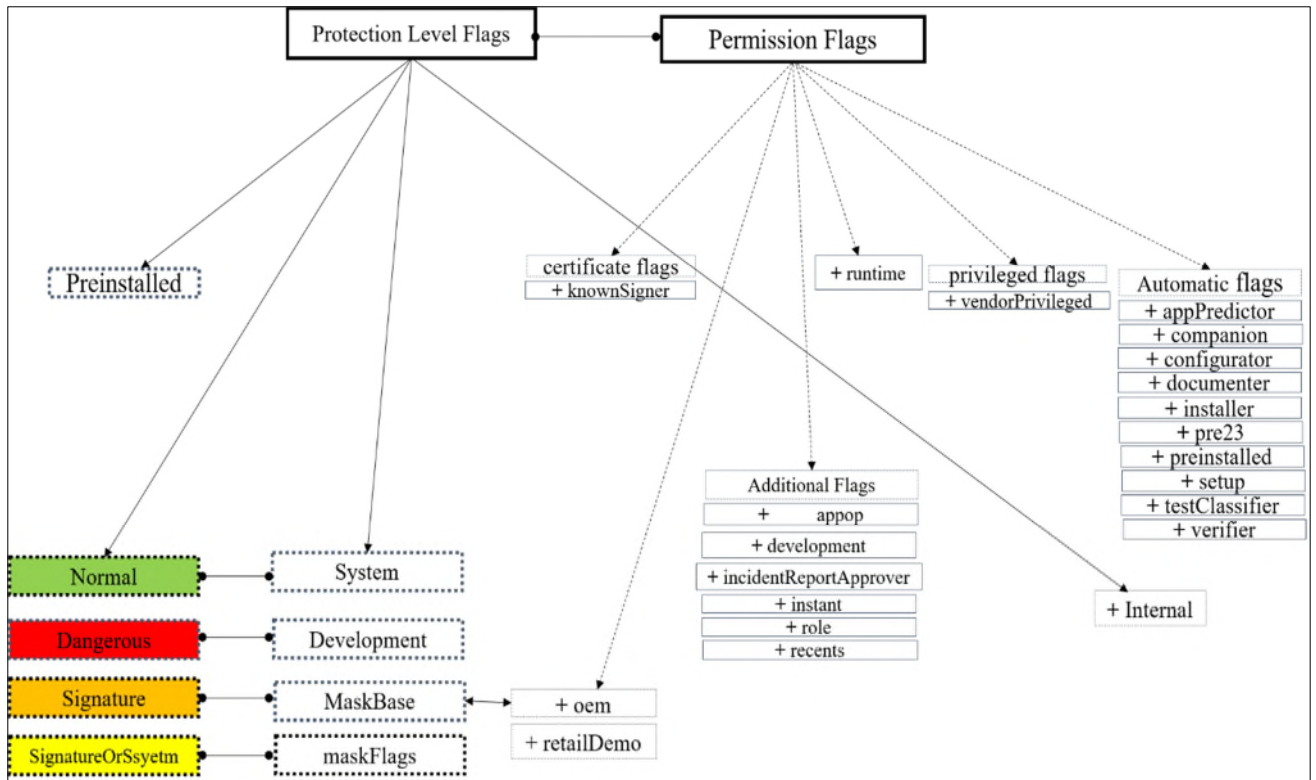


Figure 7-1 Protection level and permission-based flags

7.1.4 API calls

API stands for Application Programming Interfaces [22], whose levels are represented by integer values that uniquely identify each Android framework version. The android operating system uses APIs to reinforce third-party applications. Through APIs, entry to the device resources and features is easily made. Application programming interfaces consist of two structural components: API library and API implementation (as shown in Figure 7-2). API library is situated in the virtual machine (VM) of the Android application. The implementation structure of the API is the system's running process. These two API structures are packed in the software development kits (SDK) of the Android platform. During the device operation process, the application library of the API calls the private interface. The private interface then invokes and initiates all the remote process calls. This helps in assigning services from the service thread during the runtime process. Critical examination of the Android API call chain can aid virtually in disclosing an application's intention. CFGs form the entire representation of the Android application.

It forms a graph consisting of finite sets of nodes (N) [23] of the documented API calls and the finite set of edges (E) [24], which link successive instructions.

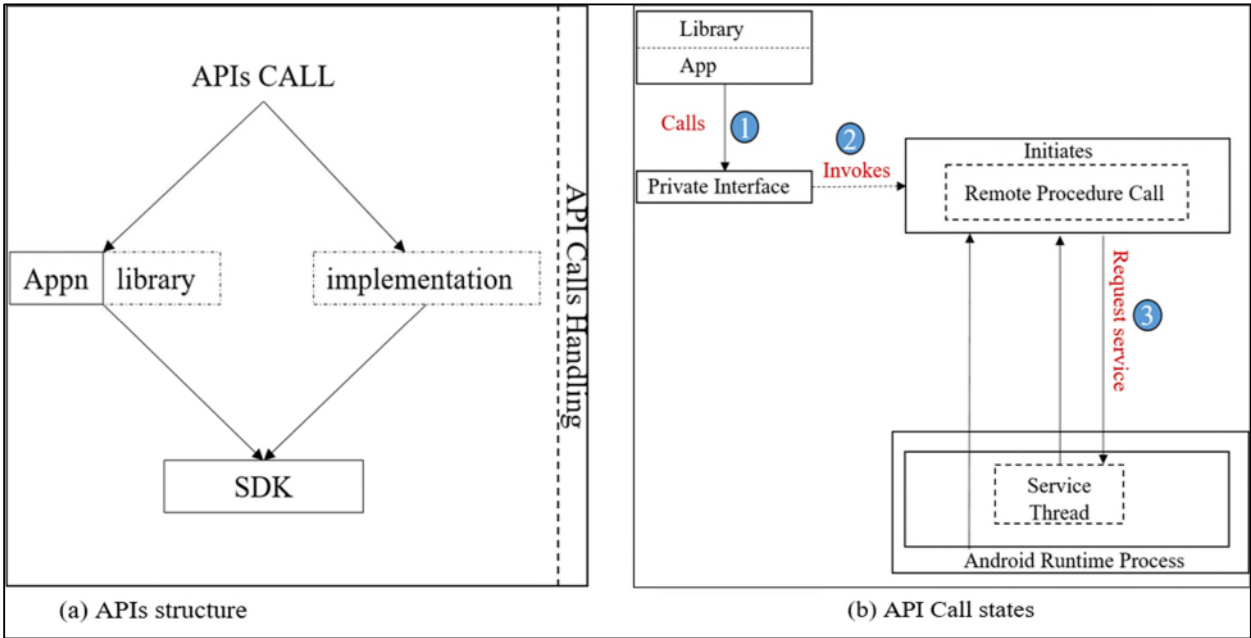


Figure 7-2 Android OS API call architectural handling process showing the connection between the APIs structure and Call states interfaces.

Control Flow Graph (CFG) is a directed graph description of how a program is controlled during execution. Straight lines, nodes, and edges are some of the graphical notations used by CFG. The ability of the CFG to link the entry block helps in summarizing the control flow. When an application has malicious intent, CFG blocks the code and makes it unreachable from appending to the Android OS or existing applications. However, adversarial examples can manipulate CFGs by modifying a sample to evade detection from a classifier or a detection engine. In the same vein, malware use code level or binary to manipulate their sample bytes during compilations. This manipulation enables benign applications to inject their block of bytes into malicious binary. According to [25], some code-level malware applies perturbation and then modifies the original code structure. When malware attack the CFG of an Android-based device, it causes structural modification of the code’s feature space.

7.2 Related work

7.2.1 Permission-based detection and feature extraction

Several studies have been carried out on Android permission requests [26], Android user privacy [27], attacks on Android and other mobile platforms [28-29], including malware attacks [30], synchronous channels attacks [31], side-channel attacks [32], runtime repackaging attacks [33], and Man-in-the-Middle attacks [34] accuracy. This section of the paper provides a brief examination of some of the related works carried out. Almomani and Al Khayer [1] discussed the Android permission systems by providing a general overview of the Android control mechanism. The study identified and grouped Android dangerous permissions based on their API levels. The research showed the significant influence of Android permissions at diverse protection levels. Zarni and Zaw [35] proposed an Android malware detection framework that collects Android applications' permissions by applying clustering algorithms. Their system monitors several permission features obtained from Android apps to determine whether the permissions are malicious or benign.

In [36], Akleyek proposed permission-based detection of Android malware using a feature selection approach based on linear regression. Using a minimum of 27 features, their research selected the most distinctive permissions to enhance the framework's performance instead of using all the permission feature vectors. In [37], McDonald et al. used four different algorithms to classify Android malicious applications as either malicious or benign based on their permission's manifest files. Using a dataset with a sample size of 5243, the best algorithm obtained a performance accuracy of 81.5% accuracy. The research concluded that hackers target manifest permission features as an attack vector to infect mobile devices. In [38], Mathur et al. proposed a detection framework for detecting Android malware using custom and native permissions features. Their findings show that the increasing number of Android permissions increases options to gain access and control over sensitive information and mobile devices.

In [39], Shahriar et al. proposed a framework for the detection of Android permissions that are anomalous. The detection system focused on the identification of relevant categories by applying a latent semantic index [40-41]. The result of the study shows that dangerous permissions constitute a security vulnerability. The understanding of permissions relationships could help in the detection of malicious applications. In [42], Li et al. proposed an Android malware detection

tool called SigPid. Their system is based on the analysis of permission usage. Using a three-level pruning approach, their study mined the permission dataset to identify significant permissions effectively and distinguishingly in malicious and benign applications. Although the study of Li et al. achieved a high detection accuracy of 93.62%, only 22 out of 135 permissions were considered. Additionally, the research did not consider the protection level or threat level of those permissions.

7.2.2 Control flow graph

In [43], Arora et al. used permission pairs to identify conspicuous permissions that lead to Android malware detection. Their study compared graphs of malicious and benign applications by mining the permission pairs. However, understanding the influence of pairing permissions is significant in malware detection using permission feature attributes. The proposed detection system, however, has some limitations, one of which is that malicious applications with single or no permission cannot be detected or analyzed. Another limitation of their research is that they did not explain which pair of permissions are predominantly present in benign or malicious applications. To overcome this challenge, an understanding of the correlation between protection levels and permission flags will help in developing a robust detection tool that could detect malware with permissions run time attributes.

In [44], Khariwal and Singh ranked intents and permission features based on their feature scores. Their research combined both permissions and intents attributes and then applied four different machine learning algorithms. Using the Bag-of-Words model [45–47], the extracted intents and permissions from the manifest file of the Android APK were embedded in the NLP [48–49]. Investigating the dynamic permissions using machine-learning methods to detect Android malware, dynamic permissions were extracted from a large number of public Android applications using J48, Naïve Bayes, and Random Forest algorithms in the research of Mahindru and Singh [50]. The results obtained by the algorithms were then compared based on their accuracy. However, their research did not demonstrate how correlated dynamics permissions are, and by this, false suggestions might be caused. The research of [51] compared two specific Android features: system calls and permission requests as an approach to malware detection. Their research identified some techniques which are essential in malware identification, and avoidance including USB examination and scanning and disabling mobile mode for automatic app download. However,

the study did not compare machine learning algorithms to determine which could be more suitable to achieve a better and more accurate detection.

7.2.3 Bayesian correlation, opcode sequence, and t-Distribution Stochastic Neighbor Embedding (t-SNE)

Bayesian correlation is one of the statistical linear mixed models that consistently produces probabilistic management of variables in a sample population. Bayes methods are used in different applications for matching functions with correlated likelihoods. The likelihoods are deployed in filtering responses used for learning training population samples [52]. Using Bayesian correlation, the research of [53] studied object localization to derive asymptotic properties of permissions with close similarities though different in their very nature. Using images as a sample population, the researchers deployed a Multiple Scale (MS) approach to determine the basis for which images with similar features can be classified using background modelling. The result of their research shows that Bayesian correlation can be effectively used for cross-correlation when using large-scale parameters for modelling relationships between variables [54]. The research of [55] used Bayesian linear regression to represent the relationship between gene particles by estimating the gene weights using correlation coefficient factors. The research used a profile dataset to model how the properties in gene A are related to gene B and vice versa.

Working on the Bayes model, the research of [56] formulated a Bayesian model using canonical analysis correlation to extract the correlation between sample data sets. In their approach, the data set decomposed into specific sets of statistical independence to help make logical inferences on other variables with similar attributes in the data set. Exploring multiple features of Android malware, the study of [57] designed a multiple feature detection framework for extraction of opcode sequence and N-Gram with symmetrical uncertainties using correlation-based models. Malware features extracted by the model were not complete in identifying some characteristics of malware. The framework was also inadequate in demonstrating the correlation that exists among those features extracted; thus, it cannot identify emerging Android malware. Using t-distribution stochastic neighbor embedding (t-SNE) for visualization, Zhang et al. [58] proposed a method for detecting malicious adverts in addresses of web pages using lexical-based features. Even though HTTPS provide authentication on the web browsers and pages to offer data secrecy [59], the study

of Barrera et al. [60] demonstrated that sophisticated malware leverage permission-based security vulnerabilities to infect Android and web-based applications.

7.3 Materials and methods

Various visualization techniques are used for analyses of pairwise similarities and relationships between variables in complex dataset distribution. Some of the virtualization techniques include Scatter plot [61], interpolation [62], and histogram [63], among others. These traditional virtualization techniques have limitations in facilitating virtualization of limited variables of data set at a time as highlighted when virtualizing their similarities as highlighted by Van [64]. This limits traditional virtualization techniques from being used on data sets with high dimensions and complexity like malware. To develop a model that generates correlation between permissions in the Android malware dataset, it is essential to automate the data analysis before virtualization.

In this research, we used the t-Distribution stochastic neighbor embedding (t-SNE) [65] to study data low-dimensional embedding and Self-Organizing Map (SOM) [66] to present the interactive view of Android permissions and identify the relationship between them in high dimensional shape. In addition to producing inputs with high dimensions, we used in our study to manage the complication of having too many points that begin to overlap, which is the same approach used in hexagonal binning [67].

In a typical permission-based architecture, numerous permissions are at the user's disposal. In our research, the processing method focused on understanding how the Android permission model is used and demonstrating how an attacker can leverage high threat levels to escalate privileges based on the permission vulnerabilities. Our methodology allows us to gain insights on how malicious use the given permission model in practice and highlights the strengths and shortcomings of the model accordingly. We note that although the case study focuses on Android, our empirical analysis is appropriate for various other Android permission-based architectures, as long as the applications are represented as a bit string of permissions.

7.3.1 Data set

Two categories of datasets were used for this study, the benign and malicious datasets. Dataset was obtained from Contagio [68], VirusShare [69], and Androzoo [70]. Though Drebin is an old dataset, we included part of the dataset obtained from the Impact cyber repository [71] to have a large dataset of both old and new permissions. Using VirusTotal scanner, 12,267 malicious applications were selected. A total of 10,837 benign applications with different categories were obtained from the Android PlayStore. This research is a continuation of the previous study on the Android malware permission request variables designed to classify Android permission requests based on protection and threat level. These data sets were chosen because they contain large app variants of malicious files that were collected from the official Google Play store and other official and legitimate alternative markets such as Slideme and legitimate repositories such as F-Droid.

To extract permissions from the selected Android applications, the Apktool was used to decompile the .apk file into different contents, including 'AndroidManifest.xml', 'Classes.dex', and the 'res'. All the applications permissions are contained in the 'AndroidManifest.xml', while the .dex has strings and Dalvik Opcodes. The most significant features were selected using information gained by extracting the similarities between sets of permission and then calculating and scoring each permission individually. Using feature encoding, the permissions were converted into binary vectors by concatenating all the features, which is the input of the model. Details of how permission features were extracted from the data set with other attributes and how they are converted into input vectors have been described in our previous research published in [72].

7.3.2 The t-Distribution Stochastic Neighbor Embedding

Take a data set D with highly dimensional input variables and a function f , which determines the distance between pairs of variables. Let $D = \{y_1, y_2, \dots, y_N\}$, and $f = e(y_i, y_j) = \|y_i, y_j\|$. To determine the K -dimensional embedding that each variable is embodied with point $\varepsilon = \{x_1, x_2, \dots, x_N\}$, x_i was expressed to belong to a set of code points expressed as $x_i \in \mathbb{R}^k$.

The joint probabilities P_{ij} that computes the similarity existing between permissions y_i and y_j is expressed as follows:

$$P_{ij} = \frac{\frac{[E(-e(y_i, y_k)^2)]}{(2\ell_i^2)}}{\sum_{k \neq i} \frac{E(-e(y_i, y_k)^2)}{(2\ell_i^2)}} \quad (7-1)$$

$$P_{ij} = \frac{P_{j|i} + P_{i|j}}{2N} \quad (7-2)$$

From Equation (7-1) above, E is the exponentiation of the f function, ℓ represents the Gaussian kernel parameter, and N is the number of variables, in our case, permissions. In our study, we defined ℓ in the dimension that $P_i = \mu$. The function μ is the perplexity of the conditional probability P , which enables optimization of the ℓ value. The optimal value of the Gaussian kernel differs at every point, and its value decreases in areas that the density of the data is higher in the data space, and it increases in regions of lower data density, respectively. As shown in Figure 7-3, the value of ℓ^2 decreases in regions where the slope is steep. This shows how the weighting matrices can be influenced when determining the correlation between two permissions in the data set. The k -dimensional embedding represented as ε measures the similarity between x_1 and x_2 pairs of points expressed as:

$$Q_{ij} = \frac{(1 + \|x_i - x_j\|^2)^{-1}}{\sum_{k \neq i} (1 + \|x_k - x_i\|^2)^{-1}} \quad (7-3)$$

From Equation (7-3) above, the student-t distribution enables the identification of dissimilar variable input y_i and y_j at the low-dimensional counterparts x_i and x_j . Applying the Kullback–Leibler principle [73], we compute the divergence existing between the Q and P express as follows:

$$K_u \ell = K_l(P\|Q) = \sum_{i \neq j} P_{ij} \log \frac{P_{ij}}{Q_{ij}} \quad (7-4)$$

The K_l in Equation (7-4) above is the normalization factor between P and Q distribution over the number of distinctive variables. To estimate the input variables similarity computed, the approximation technique of Barnes–Hut cited in [74] was applied. This technique also helps to avoid embedding quality from being negatively impacted while locating the nearest neighbors for every N number of input variables and enables seen the relationship between two dimensions with size and colour (see Figure 7-4) like bubble chat [75]. By applying the approximation technique, Equation (7-1) was reformulated as:

$$P_{ij} = \begin{cases} \frac{\frac{[E(-e(y_i, y_k)^2)]}{(2\ell_i^2)}}{\sum_{k \in N_i} E(-e(y_i, y_k)^2) / (2\ell_i)}, & \text{Provided } j \text{ belongs to } N_i \end{cases} \quad (7-5)$$

Equation (7-5) forms the vantage point tree VP-tree [76-77] for retrieving permission features from the data set without necessarily being in a high dimensional variable space based on the $e(y_i, y_j) = \|y_i, y_j\|$ metric function. The VP-tree technique enables our method to be applied on data sets whose $x_i \in \mathbb{R}^D$ vectors have input data of high dimensions.

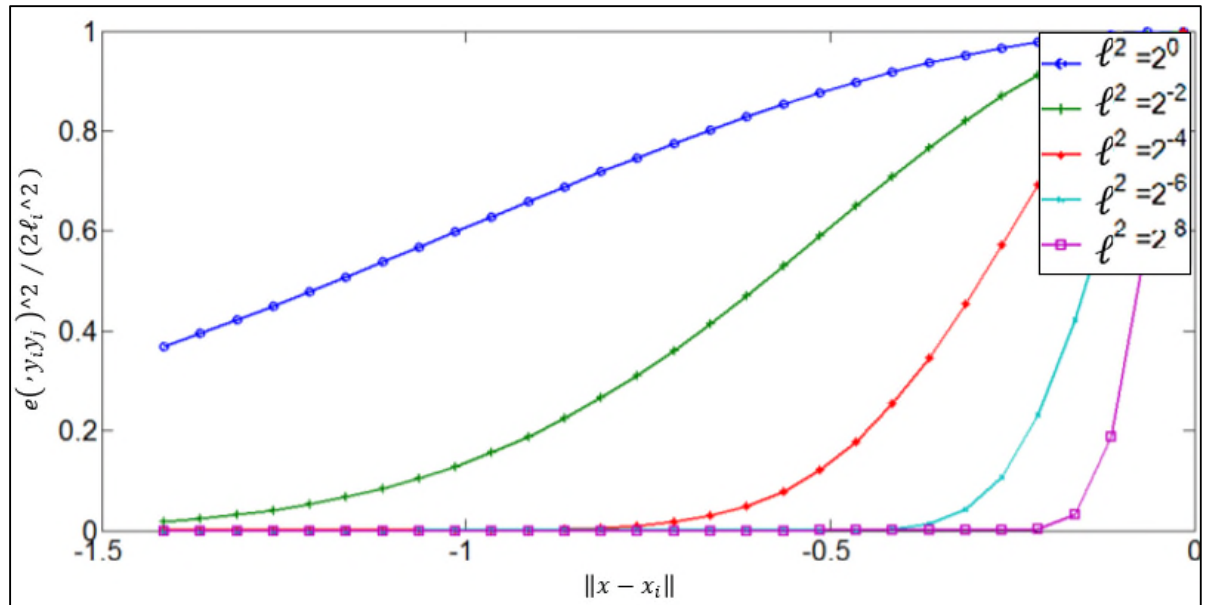


Figure 7-3 Gaussian Kernel optimisation values.

The optimal value of the Gaussian kernel differs at every point, and its value decreases in areas that the density of the data is higher in the data space, and it increases in regions of lower data density, respectively.

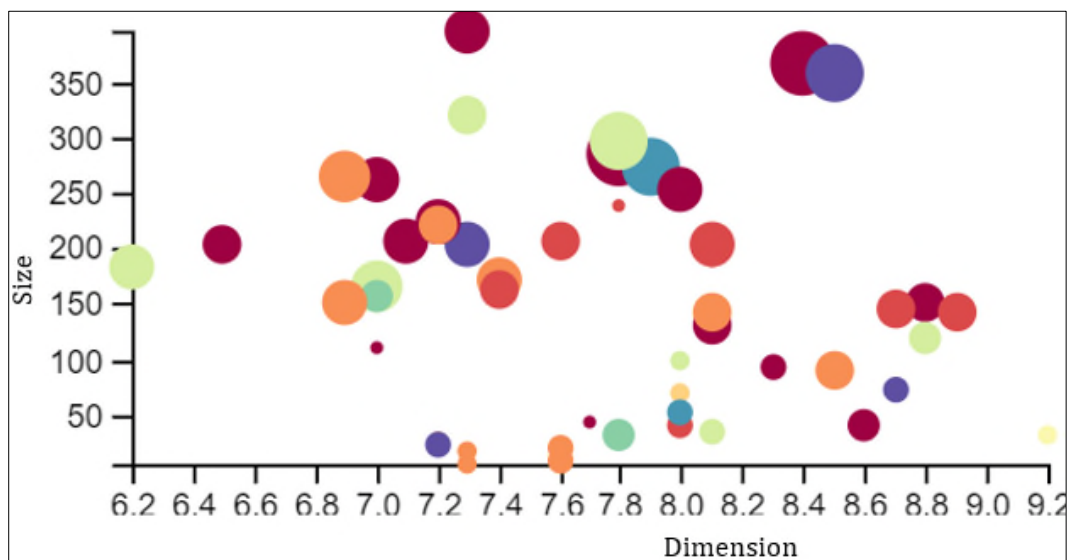


Figure 7-4 Points correlation and proportions

7.3.3 The Self-Organising Map (SOM)

As highlighted in section 7.3, SOM is an important tool for creating and representing highly dimensional input space, which binds correlated variable densities and displays them using colours or the variable region. Another advantage of this algorithm is that it can encapsulate complex data and still maintain the topographical attributes of the data in the input vector space [78]. Consider a neural network of b weights consisting of y input correlating to an Android app having its permissions represented by a binary string $\{0,1\}$. Let g represent the matching neuron; then, the Euclidian distance existing between y and b is expressed as:

$$e_d = \|y - b_g\| = \min_i \{\|y - b_i\|\} \quad (7-6)$$

$$b_i(h + 1) = b_i(h) + r_{gi}(h)[y(h) - b_i(h)] \quad (7-7)$$

Equation (7-7) was achieved by adjusting matching neuron g in the neighbor neuron index i and the kernel r_{gi} and time h . The function r_{gi} is a function with reference to the Euclidian distance and time. Given the learning rate of the model α and the location of the two-dimensional network neuron (h) , equation (7-7) becomes:

$$r_{gi}(h) = r(\|l_g - l_i\|, h)\alpha(h) \quad (7-8)$$

$$r_{gi} = E\left(-\frac{(\|l_g - l_i\|^2)}{(2\ell^2(h))}\right)\alpha(h)$$

where ℓ is the Gaussian kernel factor (Section 7.1, equation 7-1). During the network training, the learning rate of 0.01 with a batch size of 8 was set on the weight decay of 0.001 at a momentum of 0.9. We set the value of $\alpha(h)$ high to enhance better transformations in the Self-Organizing Map weight points (Figure 7-5) and the U-matrix (Figure 7-6) of the permissions. However, the limitation of setting a high value of the learning rate is that changes in the network neurons were not incremented significantly. Consequently, the training parameters must be tested using exploratory analysis before permission can be visibly visualized.

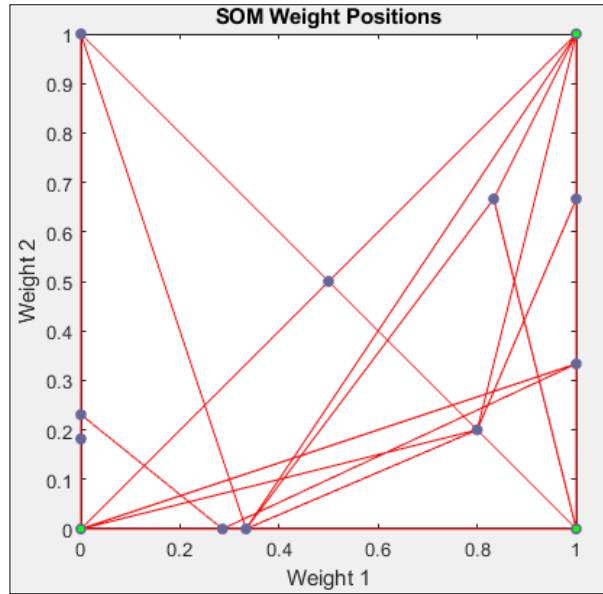


Figure 7-5 SOM weight positions of the permission input variables

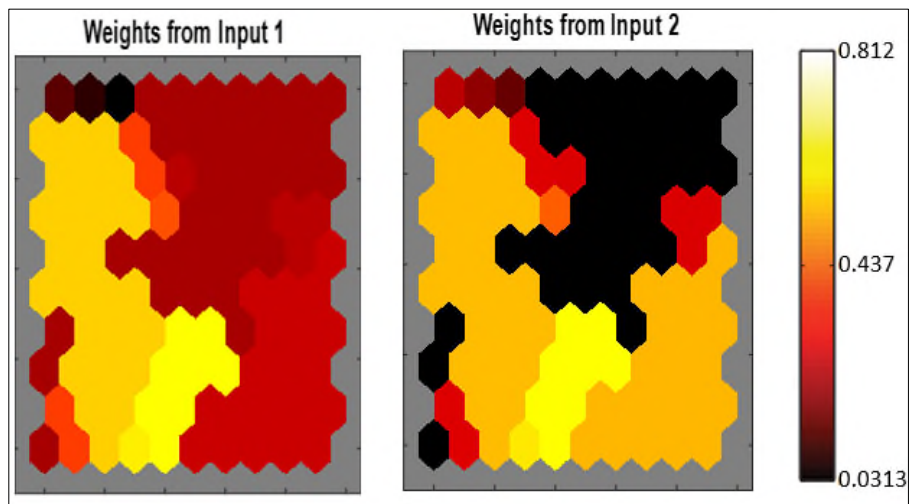


Figure 7-6 SOM U-matrix for Android permissions. The hexagons indicate spaces between network neurons with a binary string $\{0, 1\}$ which forms the Euclidean distance

As shown in figure 7-6, there are some regions (in the hexagon upper region) that are darker than others, while others are sparsely populated involving. During the inspection, we discovered that permissions presented in the darker regions are those that use traditional mobile features such as text messaging and calls, among others. This shows that those regions have related input patterns implying that Android apps in the same category with permissions in the same region do not have necessarily behave the same and request the same permissions. It then means that Android apps in

the same cluster can request similar permissions. It is then significant to understand applications and permission clusters by performing exploratory plane analysis to determine Android permission correlation and how they are frequently requested.

7.3.4 Exploratory factor analysis

Determining the correlation between permissions in complex data sets such as Android malware is challenging. A total of 12,267 malicious and 10,837 benign applications were used for our study. Exploratory factor analysis is a great simulation technique in reducing multivariate data dimensionality to identify minute factors that explain the underlying correlation between variables. This technique was applied in the research of [79] to analyze the dataset. We apply the equamax exploratory factor analysis rotation method to reduce the number of independent k factors based on their eigenvalues (see Figure 7-7) that could be obtained from many correlated permissions and to test for possible permission correlation using correlation methods (Pearson, Spearman, and Kendall correlation). Factor analysis technique enhances establishment of the best portion of permissions variables interrelationship among factors. Using Kaiser–Meyer–Olkin’s approach [80], we extracted three factors by MSA extraction procedure. The factors in the structure matrix of the selected variables using the equamax rotation method (see Table 7-2) explain the correlation among the permission variables. Each of the factors (factor 1, factor 2, and factor 3) was represented by circles, and the permission variables selected were represented by boxes in the path diagram (see Figure 7-8).

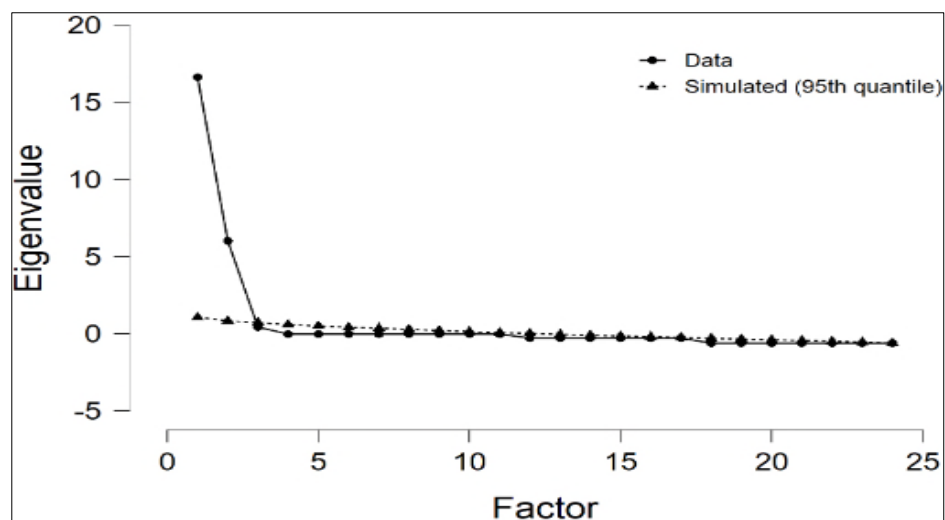


Figure 7-7 The Scree plot shows how much there is variance in the dataset by eigenvalues in the three factor

Table 7-2 Structure Matrix of some selected variables using equamax rotation method

Permissions	Factor 1	Factor 2	Factor 3
WRITE_SMS	0.972	0.077	−0.220
READ_SMS	−0.252	0.967	
SEND_SMS	0.934	−0.302	0.191
RECEIVE_SMS	0.972	0.077	−0.220
RECEIVE_WAP_PUSH	−0.252	0.967	
CALL_PHONE	0.934	−0.302	0.191
READ_PHONE_STATE	−0.252	0.967	
READ_CALL_LOG	0.934	−0.302	0.191
WRITE_CALL_LOG	0.972	0.077	−0.220
ADD_VOICE_MAIL	−0.252	0.967	
USE_SIP	0.934	−0.302	0.191
PROCESS_OUTGOING_CALLS	0.972	0.077	−0.220
GET_ACCOUNTS	−0.252	0.967	
READ_ACCOUNTS	0.934	−0.302	0.191
WRITE_ACCOUNTS	0.972	0.077	−0.220
ACCESS_CAMERA	−0.252	0.967	
READ_EXTERNAL_STORAGE	0.934	−0.302	0.191
WRITE_EXTERNAL_STORAGE	0.934	−0.302	0.191
READ_CALENDAR	0.934	−0.302	0.191
ACCESS_COARSE_LOCATION	−0.252	0.967	
ACCESS_FINE_LOCATION	0.972	0.077	−0.220
RECORD_AUDIO	0.934	−0.302	0.191
WRITE_CALENDAR	0.972	0.077	−0.220

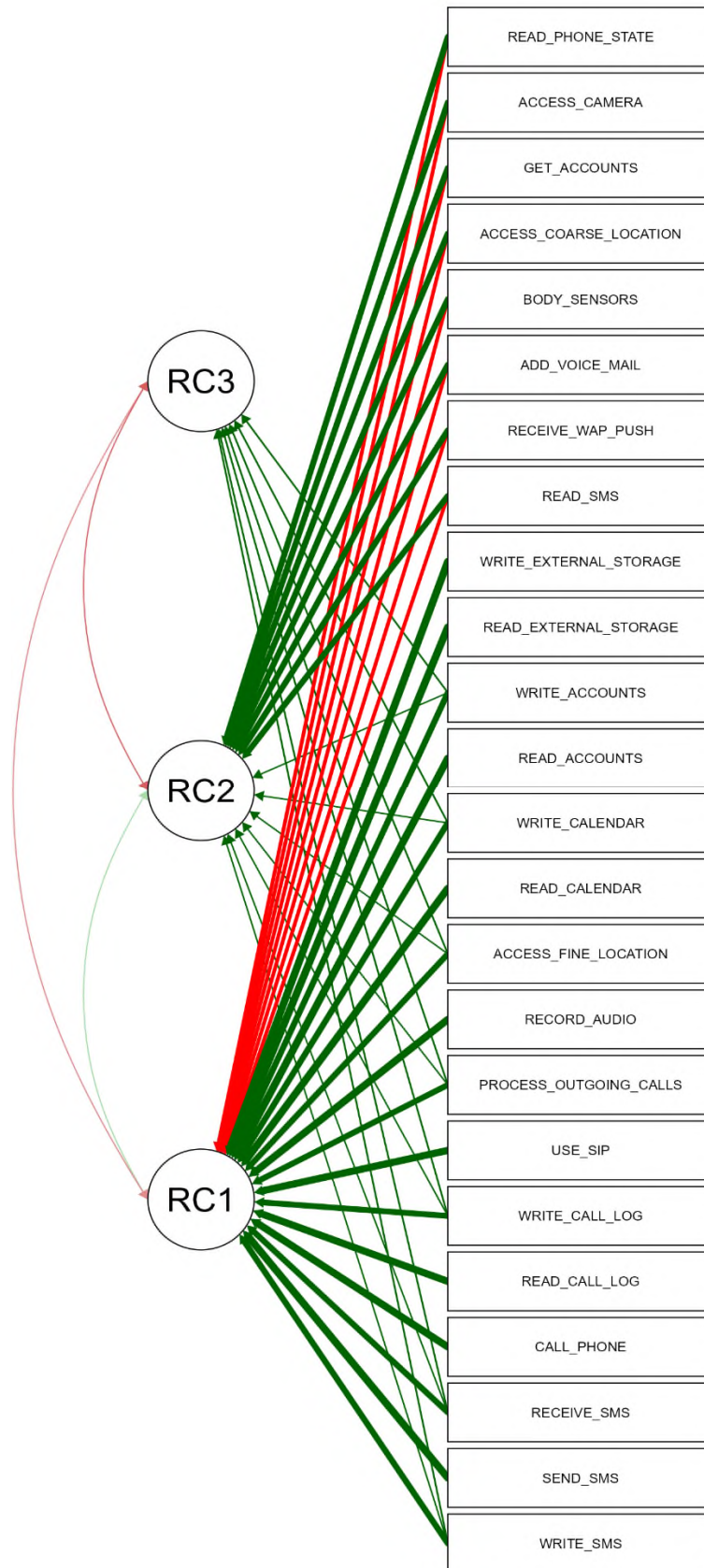


Figure 7-8 Exploratory analysis path diagram of the correlation factors and the permission variables

Figure 7-8 shows the cursors from the factors pointing to the permission variables signify how a factor loads on the permission variable. When there is a positive loading, the model represents it with green arrows; otherwise, red arrows. The dimension of the loading is determined by the size of the arrow. This means that when loading is high, the arrows become wider; when loading is low, the arrows become narrower. Accordingly, `record_audio`, `read_calendar`, `write_external_storage`, `read_external_storage`, `read_accounts`, `use_sip`, `read_call_log`, `call_phone`, and `send_sms` are associated with factor 1. `Read_sms`, `receive_wap_push`, `read_phone_state`, `add_voice_mail`, `get_accounts`, `access_camera`, and `access_coarse_location` are associated with factor 2. Lastly, `write_calender`, `access_fine_loaction`, `write_account`, `process_outgoing_calls`, `write_call_logs`, `receive_sms`, and `write_sms` are associated with factor 3. We observed that though there is a clear association of specific variables to specific factors, some of the variables are partly associated with more than two factors.

Figure 7-9 shows the exploratory factor analysis visualizing some traditional permissions frequently requested by Android applications. The result shows that most permissions in the large subcategory were requested by very few Android applications, while frequently used permissions were those in the small subcategory. With regards to state of the art, this signifies that there is no adequate expressiveness from permissions that are requested frequently. We infer that those not frequently requested could be disintegrated into the common class. As a result, we suggest adding finer granularity as a security approach for Android permissions that are frequently requested by applications will enhance this expressiveness and enhance Android security, especially when combined with the ones occasionally requested.

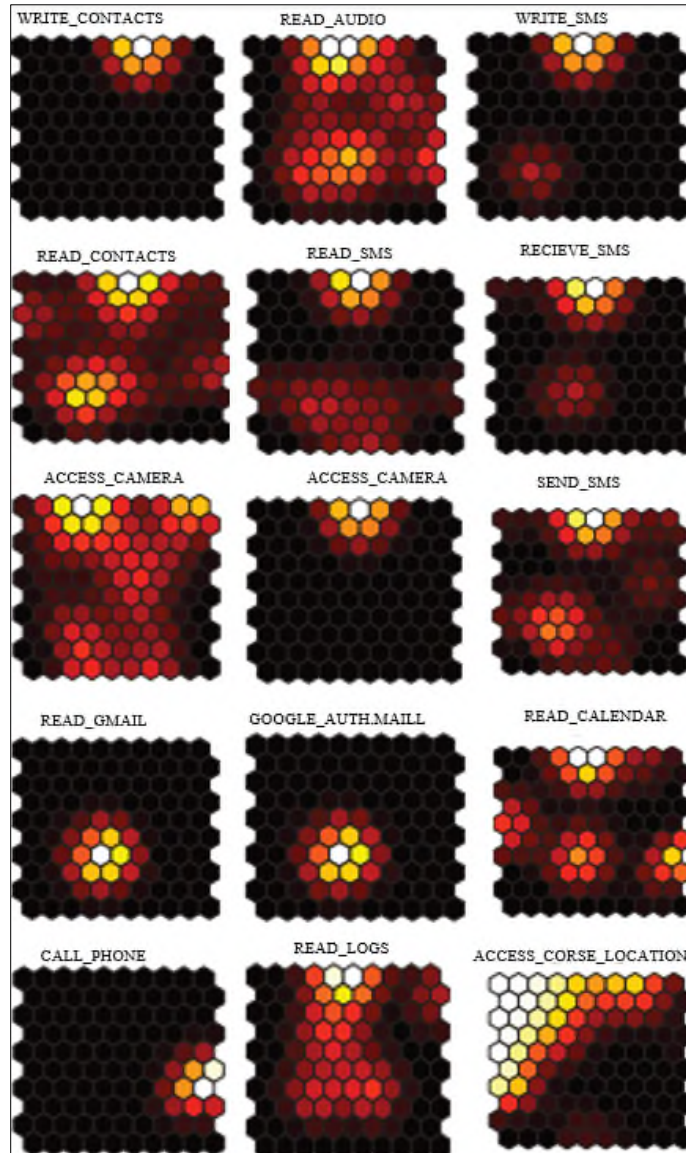


Figure 7-9 Exploratory visualisation of permission

7.3.5 Correlation between Android permissions

According to [81], when modelling relationships between variables of interest, Bayesian inference is the best starting point to formulate the prior distribution of such a model. Bayesian prior distribution enables quantization of the parameter values before sample size observation. It reflects the observed attributes of the p -values and gives more information about the data set central tendency that forms parameter estimation points. Posterior differences quantify the parameter estimation uncertainties, which indicates that the uncertainty increases with an increase in the

dispersion. However, it is not every correlation coefficient that could determine the posterior distribution from sample sizes as identified by Tan et al. [82].

In this subsection, we aim to represent the interaction between permission variables as a factor of linear combinations of threat and permission levels. Let X represent a permission variable relating to whether the threat level in the permission variable is high ($X = 1$) or low ($X = 0$). Additionally, let $\emptyset$ be the parameter symbolizing the probability that $X = 1$, that is, the threat level of the dangerous permission is high. Going by the Bayesian correlation approach, we compute and express our confidence in the probability α using the posterior distribution on parameter $\emptyset$. Following the Bayesian correlation approach, we assume that the parameter $\emptyset$ obeys the generalized hyperparameters Beta distribution [83] of the prior expectation and variance control of the $\emptyset$ prior. Let β represent Beta consisting of the prior expectation e and variance control g of the prior given as:

$$\beta(e, g) \quad (7-9)$$

Using empirical evidence [84-85], consider $\bar{e}$ to be the empirical evidence of dangerous permission in a malicious file. Let $\bar{e} = 1$ if the threat level of any dangerous permission is high in the data set, and $\bar{e} = 0$ contrarily. If no benign permission variable is present in the list of permission features, then no threat level will be observed. Otherwise, it will be detected by the probability α . The evidence for the presence of a high threat level in the model is expressed as H_1 , otherwise H_0 . The evidence is captured by delta β , which depends on the decision. Let $\alpha \in \mathbb{R}^+$ and $\beta \in \mathbb{R}^+$ be the parameters for the probability density function, which supports $x \in [0,1]$. The moments of the empirical evidence $\bar{e}$ and variance σ^2 for each permission variable obtained from the dataset expressed as:

$$\bar{e}(X) = \alpha(\alpha + \beta)^{-1} \quad (7-10)$$

$$\sigma^2(X) = \frac{\alpha\beta}{(\alpha + \beta + 1)^2} \quad (7-11)$$

To assess the goodness of fit of the cumulative distribution function of a permission variable, we applied Cramer–von Mises statistical parameterization approach [86]. In estimating the parameters, the range of x was set from 0 to 1 $x \in [0,1]$ with a confidence interval of 95%. The range is the highlighted interval for both the density and the probability (as shown in figure 7-10).

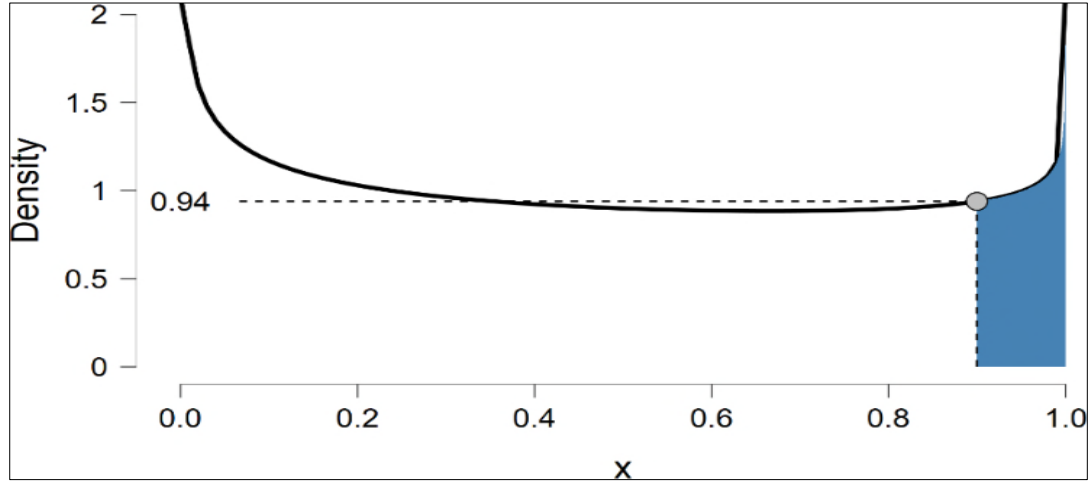


Figure 7-10 Density plot for random permission variable.

The y-axis displays the value of the density function for a particular value of the permission variable. The dotted line represents the density, while the blue-coloured region is the probabilistic region. In the analysis, we chose to reflect on the risk permission request poses on mobile devices, particularly the Android platform. We focus on analysing the relationship between the threat and protection level. However, we are uncertain of the probability that a permission request has a high threat level or low protection level. The decision to set α over the parameters was based on the robust Bayesian approach [87] of analysing uncertainties in variables correlation, assuming that the variance is homogenous in the variable distribution. Given that the variance treatment is equal:

$$\sigma^2_1 = \sigma^2_2 = \dots = \sigma^2_k, \quad \sigma^2_i \neq \sigma^2_j \quad (7-12)$$

In the above equation, k is the number of permission groups in the sample distribution and i and j represent at least one pair. To check for sample deviation from the normal distribution, we applied Bartlett's test assumption check [88]. Let σ_p^2 represent the pooled variance expressed as:

$$\sigma_p^2 = \frac{\sum_{i=1}^k (N_i - 1) \sigma_i^2}{(N - k)} \quad (7-13)$$

$$\chi^2 = \frac{(N - k) \ln(\sigma_p^2) - \sum_{i=1}^k (n_i - 1) \ln(\sigma_i^2)}{1 + \frac{1}{3(k - 1)} \left(\sum_{i=1}^k \left(\frac{1}{n_{i-1}} \right) - \left(\frac{1}{N - k} \right) \right)} \quad (7-14)$$

Equation (7-14) represents Bartlett's test. σ_p^2 represent the pooled variance, σ_i^2 is the variance of the i^{th} permissions groups, N is the permission sample size, n_i is the permission sample size in the i^{th} variable group, and k is the number of factors with k -levels. The result of Bartlett's test shows the interaction of the effects of permission sample size and other parameters (see Table 7-3).

Table 7-3 Interaction effects of the test statistics using MANOVA

n	χ^2	k	σ^2
$n1$	15.925	2.130	0.708
$n2$	16.002	3.403	0.611
$n3$	16.201	3.342	0.579
$n4$	15.925	2.133	0.731
$n5$	16.002	2.421	0.816
$n6$	15.001	3.342	0.903
$n7$	15.521	2.530	0.881
$n8$	15.106	3.501	0.908
$n9$	17.067	2.367	0.736

If the correlation that exists between Android permissions is represented as ρ , consider permission variables with respective prior odds represented as $p(u_0)|p(u_1)$ in the dataset represented as $D = \{y_1, y_2, \dots, y_N\}$. The model posterior odds are expressed as:

$$\frac{p(u_0|D)}{p(u_1|D)} = \frac{p(D|u_0)}{p(D|u_1)} \times \frac{p(u_0)}{p(u_1)} \quad (7-15)$$

where u_0 and u_1 represents instantiations for the null hypothesis (H_0) and alternative hypothesis (H_1), respectively. From Equation (18), the function $\frac{p(D|u_0)}{p(D|u_1)}$ forms the Bayes factor. This is the proportion of the likelihoods of the posterior distribution. However, we represented the function

(Bayes factor) in our model as Bf_n where n takes values of $_{10}$ and $_{01}$ of the permission sample size in the i^{th} variable group expressed as:

$$Bf_n = \frac{p(u_0|D)}{p(u_1|D)} \times \frac{p(u_1)}{p(u_0)} \quad (7-16)$$

These factors represent the two sides of the research hypothesis at a stretched beta prior value range of -1 to 1 using density ratio. Our Bayesian correlation model was formulated using Pearson's correlation coefficient represented as ρ . At $\rho = 0$ under Bf_{01} and posterior under H_1 expressed as:

$$Bf_{01} = \frac{p(D|H_0)}{p(D|H_1)} = \frac{p(\rho = 0|D, H_1)}{p(\rho = 0|H_1)} \quad (7-17)$$

$$Bf_{10} = \frac{p(D|H_1)}{p(D|H_0)} = \frac{p(\rho = 1|D, H_0)}{p(\rho = 1|H_0)} \quad (7-18)$$

In the alternative hypothesis, u_1 instantiates the presence of correlation between permission requests variables represented as $H_1: \rho \neq 0$. Using Jeffrey's prior [89], we assign prior distribution correlation under the null hypothesis to take uniform default values between the range -1 to $+1$ as a default ρ . This was relative to the alternative hypothesis using two population correlation coefficients, namely Pearson's rho and Kendall's tau. The two correlation coefficients were used for the posterior and prior comparison of our model (as shown in figure 7-11).

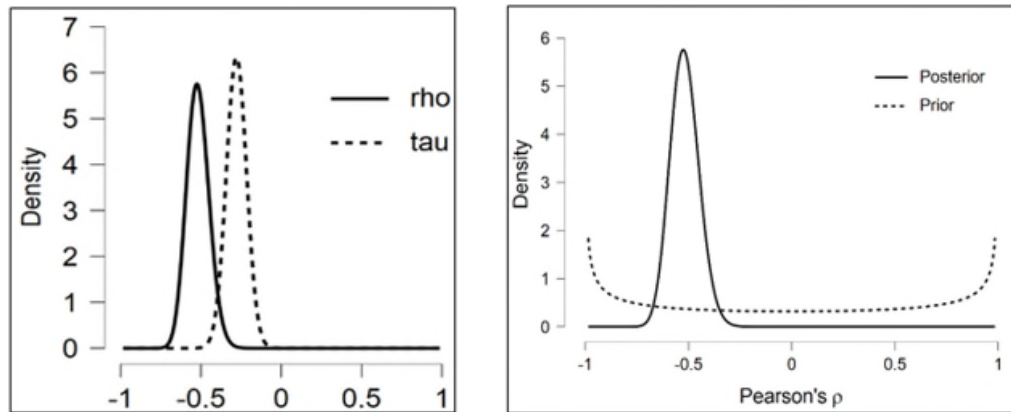


Figure 7-11 Sequential analysis of the empirical evidence of dangerous permission with high threat levels

7.3.5.1 Pearson correlation

The non-parametric function in our study helps to measure how permission request variables correlate to each other. Given x and y as permission variables contained in the Android malware dataset, the Pearson correlation is expressed as:

$$r = \frac{n\sum x_i + y_i - \sum x_i + \sum y_i}{\sqrt{n\sum x_i^2 - (\sum x_i)^2} \sqrt{n\sum y_i^2 - (\sum y_i)^2}} \quad (7-19)$$

The n represents the sample size in the dataset distribution. In determining r , we assumed that all the permission request variables in the dataset were normally and equally distributed according to the principle of variable homoscedasticity [90–92]. Otherwise, the principle of variable heterogeneity [93] sets in. Let τ represent the heterogeneity that occurs in the Android data set D . We represent the group-level effect size with μ . The Bayes factor Bf_n then propagates a heterogeneity factor written as Bf_{rf} and Bf_{fr} under prior H_1 and random factor H_1 (as shown in figure 7-12). At this instance, the Bf_{10} gives the Bayesian inference while Bf_{01} is automatically set to zero. This helps in accepting or rejecting the null hypothesis. We assumed that when heterogeneity occurs in the permission distribution, the Bayes factor becomes heterogeneous for H_1 . Table 7-4 shows the posterior estimate of the model.

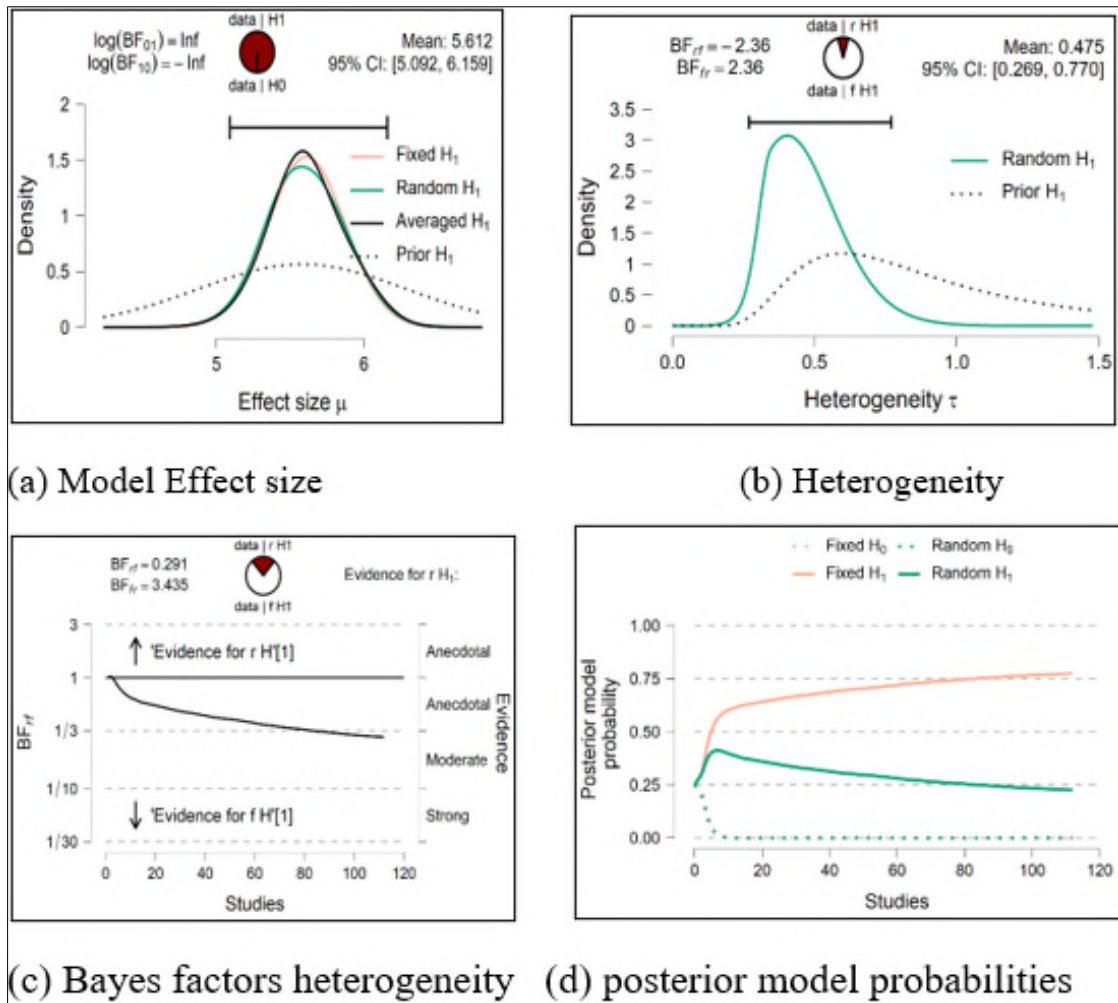


Figure 7-12 Flag supported correlation when permission distribution is heterogeneous in the sample distribution

Table 7-4 Posterior estimate of the model

Effects		Mean	SD	Lower	Upper	BF ₁₀
Fixed effects	μ	5.585	0.274	5.030	6.117	2.8×10^{84}
Random effects	μ	5.588	0.276	5.049	6.124	7.2×10^{51}
	τ	0.209	0.163	0.010	0.601	0.291
Averaged	μ^b	5.586	0.273	5.045	6.119	∞
	τ^c					0.291

From Table 7-4, μ is the group-level effect size which forms the mean in the distribution. ^a Bayes factor of the random effects H_1 over the fixed effects H_1 . ^b Posterior estimates are based on the models that assume an effect to be present. The Bayes factor is based on all four models: fixed effects H_1 and random effects H_1 over the fixed effects H_0 and random effects H_0 . ^c Model averaged posterior estimates for τ . From the data set variables, selecting four of the Android permissions

requests: READ_SMS, CALL_PHONE, RECEIVE_SMS, and WRITE_SMS of n total sample population of the permission variables, the Pearson r was determined using the p -value and the VS-MPR (<https://doi.org/10.17862/cranfield.rd.13363322.v1> (accessed on 2nd, August 2021)). As shown demonstrated in Figure 7-13, Pearson r takes a value range between -1 and 1 . The r value measures the potency of the correlation between the data variables under examination. Using the rule of Thumb postulated by Guildford [94] to determine the degree of correlation between permission requests, our research states that if the value of r is greater than 0 , there is a positive correlation between the Android permission variables.

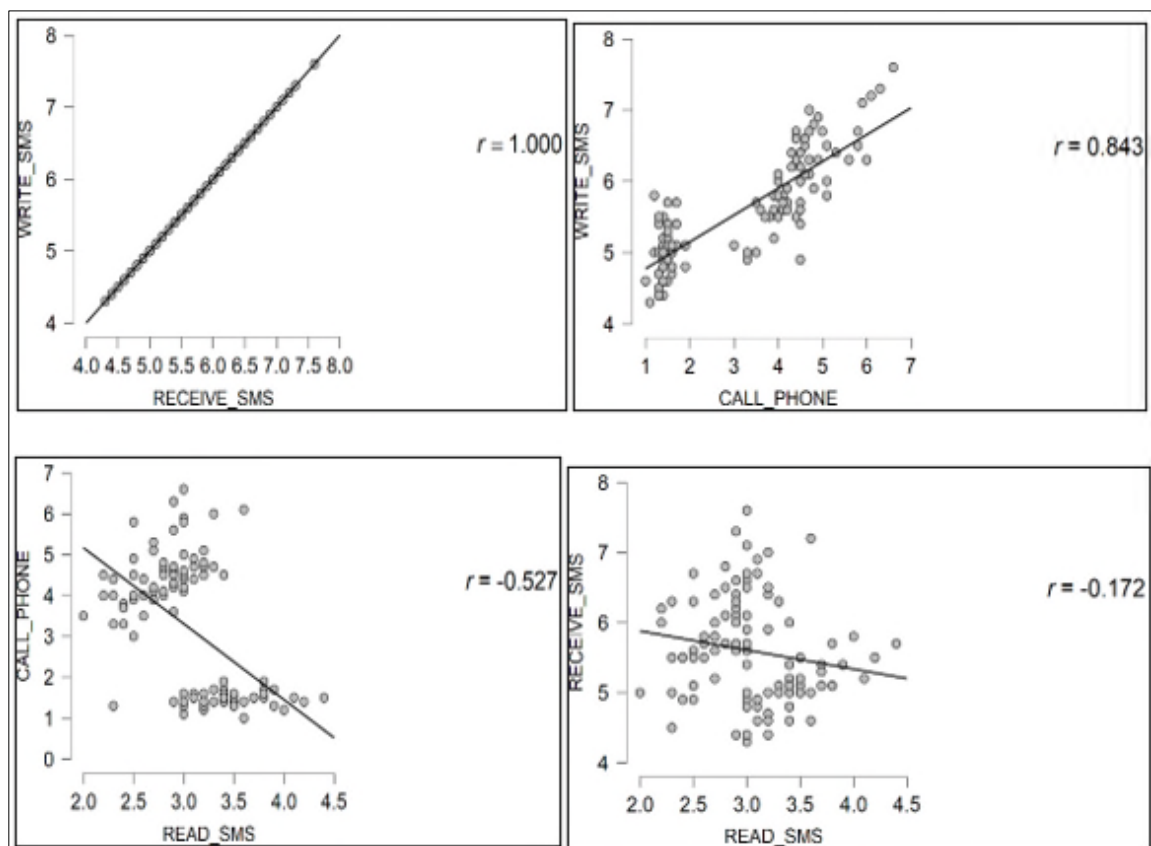


Figure 7-13 Pearson's correlation coefficient for some selected permission variables

If the value of r is less than 0 , then there is a negative association that exists between the threat level and the protection level. However, if r turns close to 0 , it implies that the correlation existing between permissions is weak. The correlation between two permissions is considered very strong if the value of r is greater than 0.7 . In a situation where the correlation between the permissions request is negatively correlated, the result of the p -value is less than 0.001 , expressed as $p < 0.1$.

7.3.5.2 Spearman Correlation

A non-parametric check measures the level of magnitude of how two or more permissions requests in the dataset are associated with one another. However, our research only focused on determining the degree of correlation between two permissions and not multi-permissions variables association. We assumed that the scores in each permission request are uniformly associated with other permission requests. Given the sample size n and the rank difference d_i between the corresponding permission variables, the Spearman correlation ρ , between two permission requests is expressed as:

$$\rho = \frac{1 - 6\sum d_i^2}{n(n^2 - 1)} \quad (7-20)$$

Let us consider some permission requests from the sample space $n = 112$. The degree of ranking between any of the two permission variables helps in ordering the observation of their correlation magnitude. The ranking order demonstrates that the higher the p -value existing between two permission requests, the lower the value of Spearman ρ and vice versa. If Spearman ρ and the p -value are inversely proportional, the correlation between the threat level and the protection level of a permission request is inversely proportional to each other (<https://doi.org/10.17862/cranfield.rd.13363322.v1> (accessed on 2nd, August 2021)). Figure 7-14 shows Spearman's rho correlation coefficients of some selected variables.

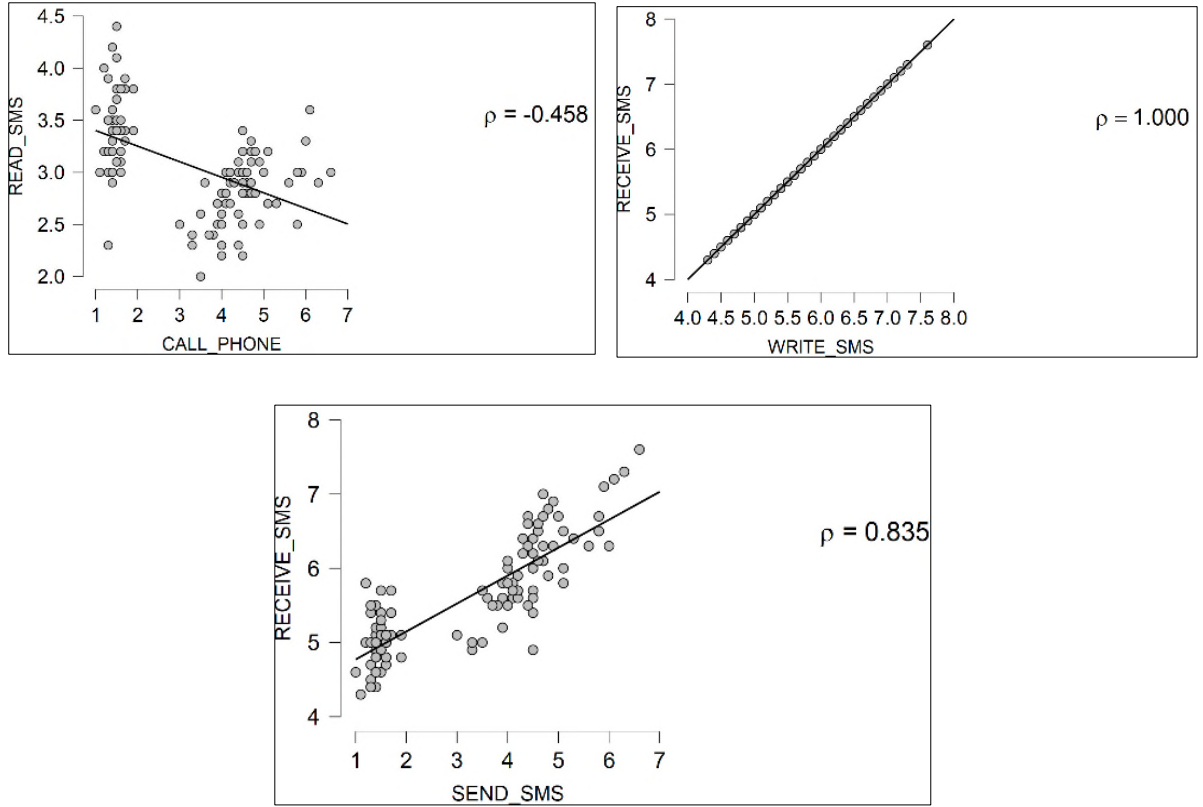


Figure 7-14 Spearman's rho correlation coefficients

7.3.5.3 Kendall Correlation

Kendall correlation measures the stability of how two permission variables depend on each other. For instance, if we consider four permission variables say: WRITE_SM, SEND_SMS, RECEIVE_SMS, and READ_SMS with a sample size n , according to [95–96], the composite number of matches between two variables can be expressed as $n(n - i) / 2$. The Kendall correlation between the variables can be expressed as:

$$\tau = \frac{n_c - n_d}{1 / 2(n - 1)} \quad (7-21)$$

where n_c and n_d represent the number of concordance and discordance, respectively. The number of concordances ensures an even ordering of permission variables while discordance orders permission variables heterogeneously or abnormally. The p -value took a range of $p < 0.05$ $p < 0.01$ $p < 0.001$. The -Sellke Maximum p -Ratio (VS-MPR) was Based on the p -value, where the

maximum possible odds in favour of H_1 over H_0 equals $1/(-e \log(p))$ for $p \leq 0.37$, as shown in Figure 7-15.

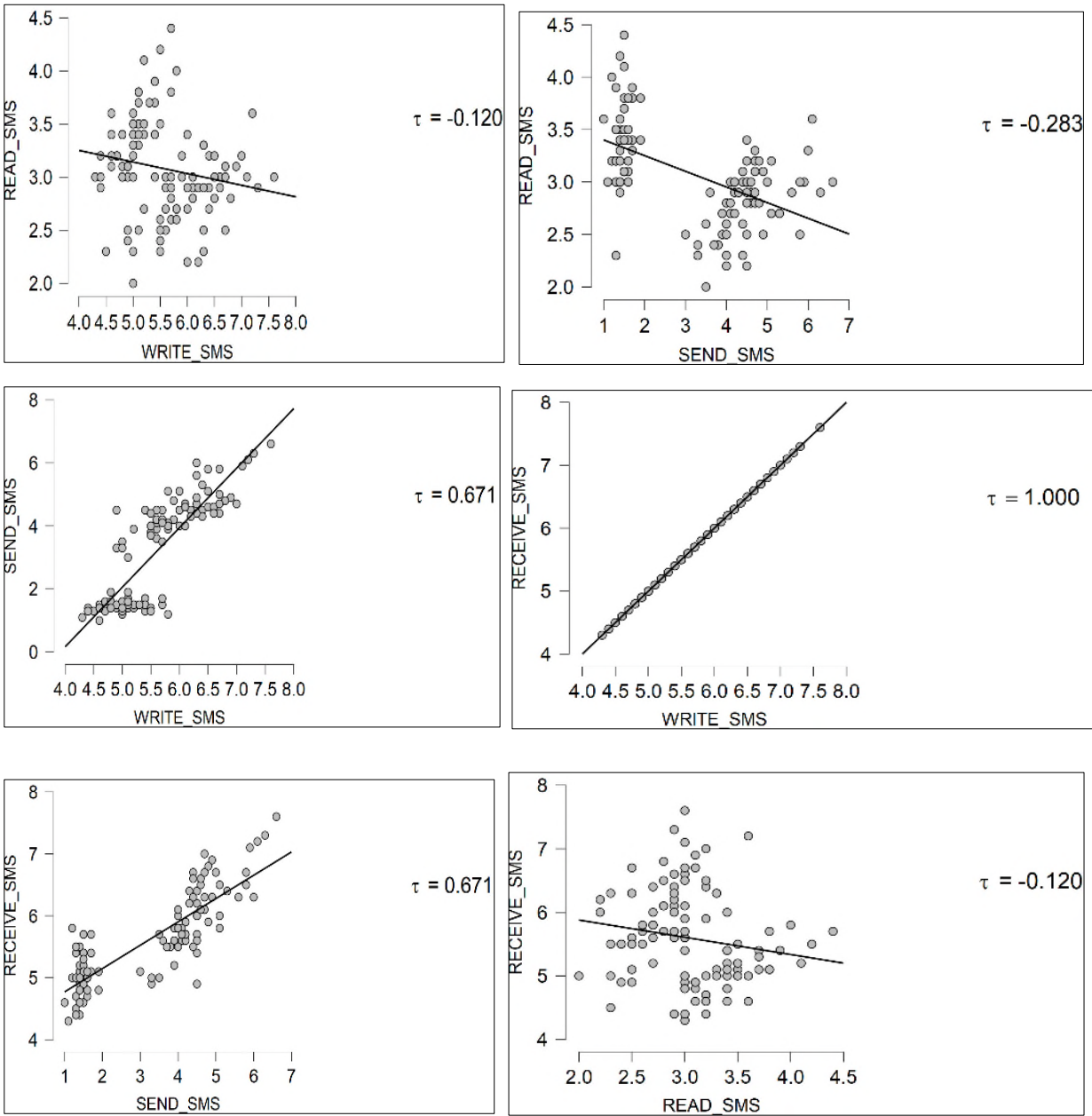


Figure 7-15 Kendall's tau-b correlation coefficient for some selected permission variables

As shown in Figure 15, it is only the correlation between RECEIVE_SMS and WRITE_SMS that has $\tau = 1$. It, therefore, means that it is only the correlation between RECEIVE_SMS and WRITE_SMS that captures the exact non-linearity that exists in all the four selected permission request variables. The stability of dependence tends to increase as the sample size increases. When

correlating a few variables, there is no significant difference in the appearance of the scatter plots of the plotted variables. However, the difference becomes clearer when the sample size increases (<https://doi.org/10.17862/cranfield.rd.13363322.v1> (accessed on 2nd, August 2021)).

7.3.6 Comparison of the correlation coefficients

This sub-section of the research paper compares the correlation coefficients under study: Pearson, Spearman, and Kendall's tau-b. The comparison aims to test the statistical significance to assess if the critical value is less than the observed value. This will help in rejecting the null hypothesis under study. During this study, we observe that the correlation line is difficult to identify at extreme negative values example, $r = -0.172$, $\rho = -0.243$, and $\tau = -0.120$. This shows that a correlation is weak in those instances. However, it becomes more visible and clearer as the correlation becomes stronger. The order of permissions requests in the correlation is not significant but provides the only evidence of association, not causation. As shown in figure 7-16, the positive values and the negative values indicate a positive and negative association between the permission requests, respectively.

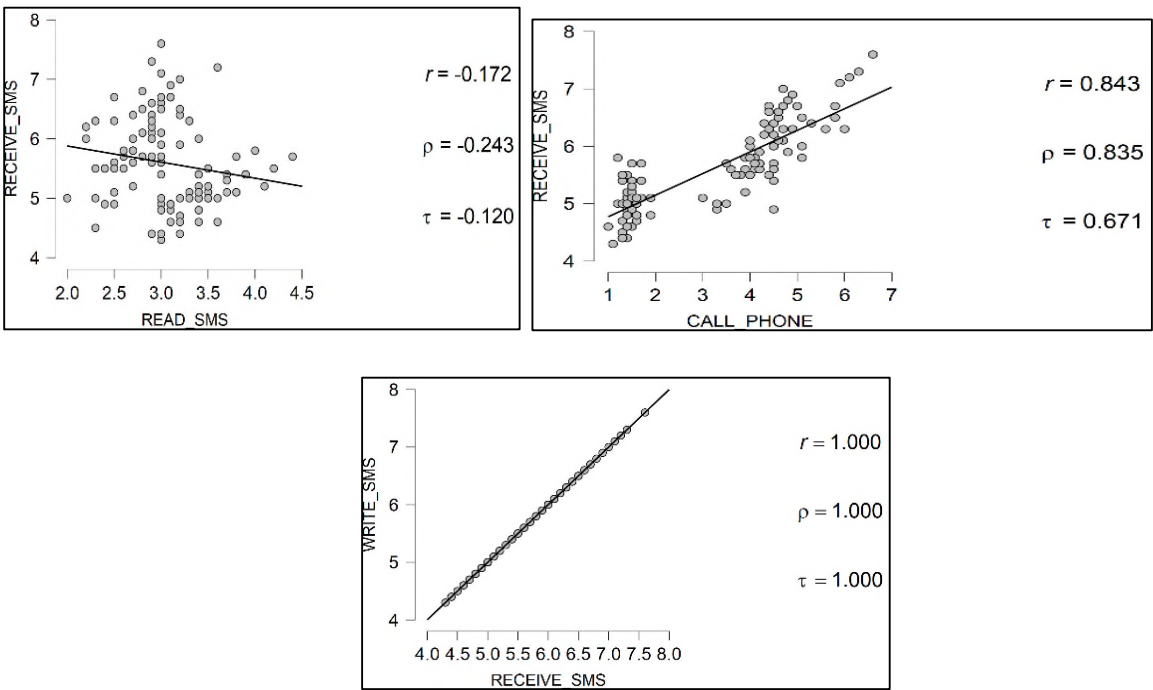


Figure 7-16 Correlation at different degrees of coefficient strengths

Extreme observations are affected and strongly impacted by the correlation coefficient. This signifies that the value of extreme values decreases as the number of samples increases. This information is necessary to help us conduct further analysis to provide additional information about the relationship between threat level and protection level of each permission request. The overall correlation table that compared the results of the three correlation coefficients generated by our model (<https://doi.org/10.17862/cranfield.rd.13363322.v1> (accessed on 2, August 2021)). The correlation heatmap for Spearman's rho, Pearson's r correlation, and Kendall's tau-b correlation heatmaps (https://doi.org/10.17862/cranfield.rd.13363322.v1_1 (accessed on 2, August 2021)).

7.4 Threat and Protection Levels Evaluation

A threat model can be defined as a well-defined description regarding the information which affects or influence an application's security. Modelling a threat level involves information capturing, organization, and analysis to facilitate decision-making concerning the security threat of an application. Modelling threat helps to identify vulnerabilities and other risks associated with an application or a system. We acknowledge that dangerous permissions as threats are hard to address apart from adjusting to their requirements. Privacy threats caused by dangerous permissions create a serious security risk to users and system developers.

This subsection of the research focuses not on developing a threat model but on determining the relationship between threat and protection level in some selected Android permission requests. Understanding the threat level of permission is significant to classify permission to be normal or dangerous. The implication is that if the permission access granted has not much negative effect on that resource; such a permission request has a low threat level while the device has high protection level at that instance. Consider a malicious Android application requesting permissions to access the camera resource of the mobile device when the user does not open a feature that necessitates using the camera, it looks so suspicious. Using Bayesian hierarchical modelling was deployed in our research to estimate the relationship between protection and threat level in each permission request by an Android malicious application. In our approach, we defined the two concepts as Level 1 for protection level and Level 2 for threat level, respectively.

Level 1: Protection level. Protection level indicates the possible risk suggested in Android permission and the process a device should adhere to when deciding whether allow an application access system's resources. In our model, it is represented as the observed data in the distribution of the permissions selected from the sample space. Let φ and ∂ denote true variance values, and φ_ε and ∂_ε represent their respective errors associated with their distribution. We then state that for each i^{th} observation taken values from 1 to N represented as $i = 1 \dots N$, their observed data values assume values with respect to their errors and variances respectively written as:

$$\hat{\varphi}_i \sim \text{Normal}(\varphi_i, \sigma^2_{\varphi_{\varepsilon i}}) \quad (7-22)$$

$$\hat{\partial}_i \sim \text{Normal}(\partial_i, \sigma^2_{\partial_{\varepsilon i}}) \quad (7-23)$$

From equations (7-23) and (7-24), $\sigma^2_{\varphi_{\varepsilon i}}$ and $\sigma^2_{\partial_{\varepsilon i}}$ represent the assumed know priori error variances for φ and ∂ . The error variances across the observation N do not assume homogeneity but each of the i^{th} observations assumed their unique individual error variances.

Level 2: Threat level. This is meant to provide a distinct indication of the probability of a malicious attack on a mobile device. In our model, threat level represents the inferred parameters for φ_i and ∂_i observations. The informed parameters assumed bivariate normal distribution with their respective means and variances. Let the mean and variance parameters be defined as Mean: $\mu \in \mathbb{R}$; Variance: $\sigma^2 \in \mathbb{R}^+$ with a support $x \in \mathbb{R}$ where $\mathbb{R}$ assumes the value $\{0,1\}$ properly written as $x \in \{0, 1\}$. The two parameters are functions of the two moments $E(X) = \mu$ and $\text{Var}(X) = \sigma^2$, respectively. Additionally, let μ_φ and μ_∂ be the mean for the inferred parameters α , which forms the effect size of the distribution given as:

$$\mu = \left(\frac{\mu_\varphi}{\mu_\partial} \right) \quad (7-24)$$

The covariance confusion matrix estimated from the prior distribution of the selected sample set is given as:

$$Cov(\varphi, \partial) = \begin{pmatrix} \frac{\sigma^2_{\varphi}}{\rho\sigma\varphi\sigma_{\partial}} & \frac{\rho\sigma\varphi\sigma_{\partial}}{\sigma^2_{\partial}} \end{pmatrix} \quad (7-25)$$

The values of the prior of μ_{φ} and μ_{∂} are set to be large $(-1, 1)$ according to Jeffreys' principles [97] to generate no uniformity for its distribution while the prior for the variance, σ_{φ} and σ_{∂} are equally set with the same dimension to prior distribution uninformative. The advantage of setting in this dimension is to enhance the automatic adjustment of ρ when a new data variability source that could result in observations uncertainty is added to it. Specifically, $\hat{\varphi}$ and $\hat{\partial}$ observations shrink in the direction of their matching class mean. In the model, if the observed parameters are highly dispersed than the inferred parameters, then permission should consider having a higher threat level than the protection level.

The relationship between level 1 (protection level) and level 2 (threat level) with data density distribution using RM factor. The result of the correlation between the observed (protection level) and the inferred (threat level) is shown in figure 7-17. This demonstrates that the posterior distribution of the observed correlation is decreased to lower values compared to the inferred correlation of the model correspondingly. Figure 7-17 shows the correlation between the represented levels estimated at φ_{ε} and ∂_{ε} , respectively. The posterior representation and the model accuracy in the estimation of the inferred and the observed parameters. The summary of the evaluation metrics for the model's precision, recall, F1 score, support, and the AUC is represented in Table 7-5.

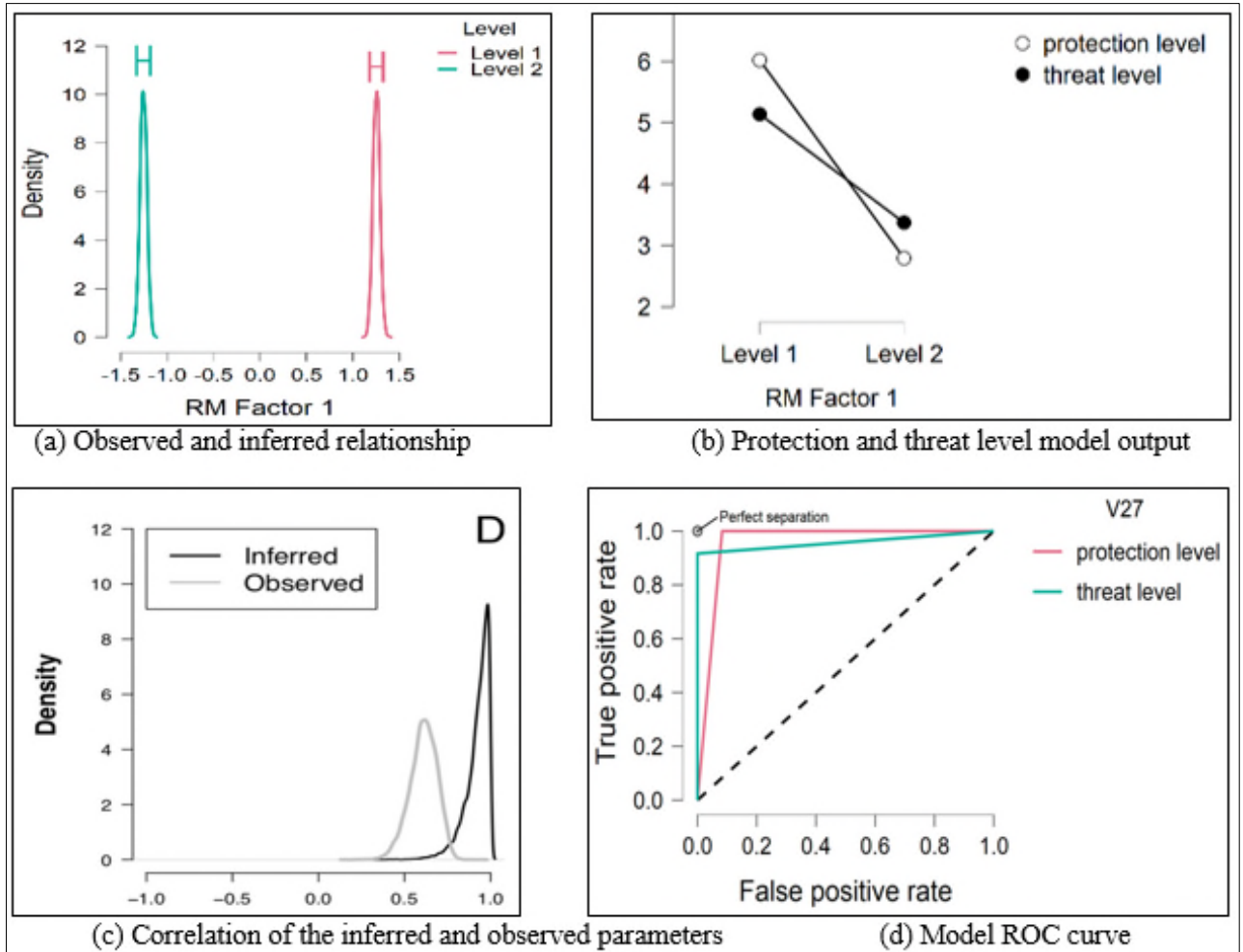


Figure 7-17 Model posterior distribution showing the correlation between the observed data (protection level) and the inferred data (threat level)

Figure 7-17a shows the observed and inferred relationship of the RM factor and the density levels as detailed in Table 7-10. Figure 7-17b shows the protection and threat level correlation model output which demonstrates how the inferred and observed parameters in figure 7-17b are related with reference to the ROC curve figure 7-17d.

Table 7-5 Evaluation metrics

	Precision	Recall	F1 Score	Support	AUC
Protection level	0.917	1.000	0.957	11	0.996
Threat level	1.000	0.909	0.952	11	0.975

Figure 7-18 is the matrix plot for correlation between protection and threat level existing between permissions. The red colour indicates the threat level while the blue indicates the protection level of individual permissions, which is distributed sparsely in an application. The denser region shows that the relative correlation between the threat and protection level between the two permissions is strong. One of the remarkable findings is that each permission has a protection and threat level. We also identified that Android permissions with the same protection level have the same threat level. This signifies that such permissions have a similar component plane in their distribution in the sample space. The details of the correlation model comparison of some selected permissions are presented in Table 7-6.

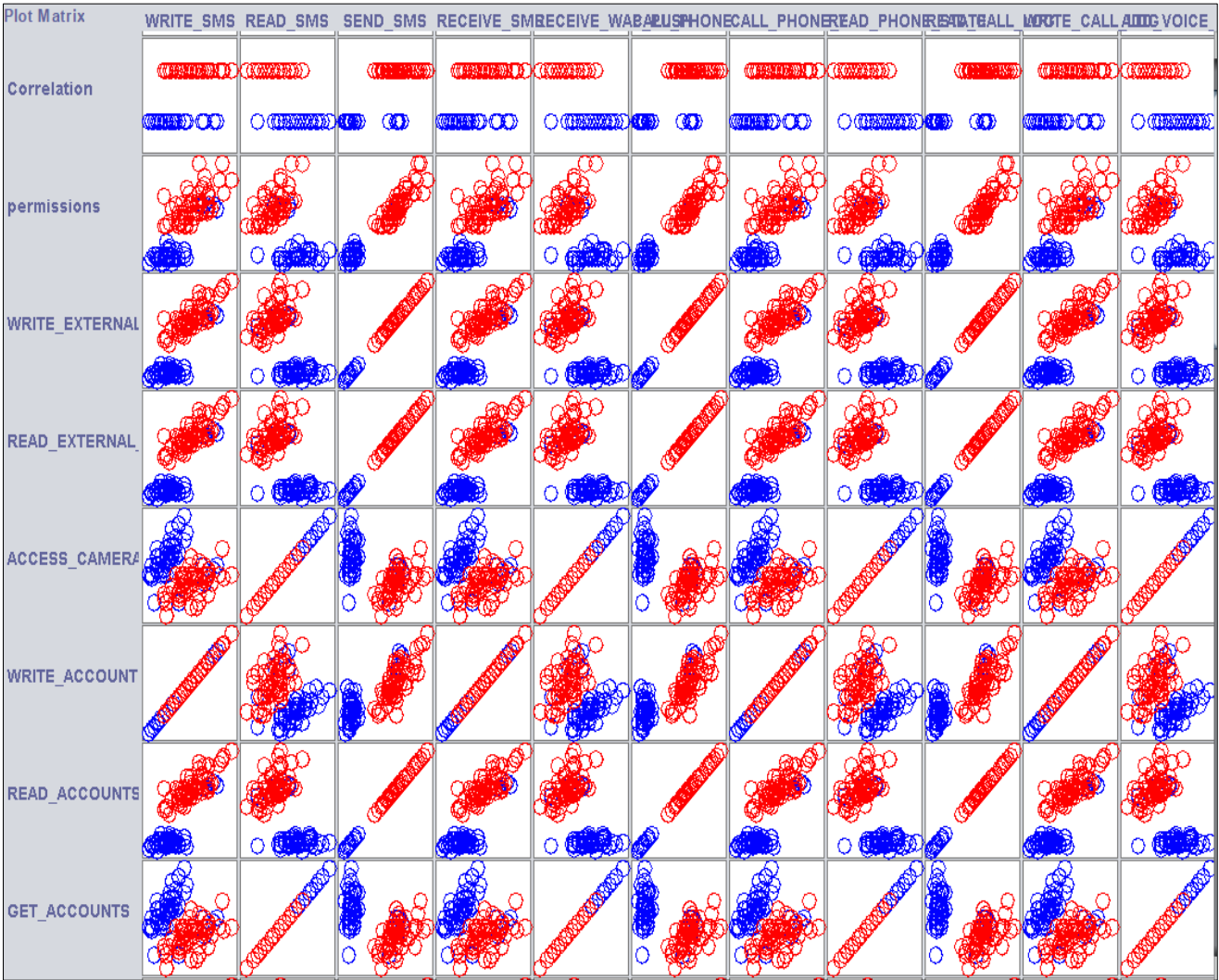


Figure 7-18 Plot Matrix showing exploratory plane analysis visualization of the correlation between threat and protection level in permissions.

The red colour in the plot matrix of the plane analysis indicates the threat level, while the blue indicates the protection level which forms the guarded region. To segregate individual permissions and visualize their threat and protection level, we adopted a minimization approach [98]. Consider two permission parameters φ and ∂ represent two different random variables. Let φ represent the observed random variable and ∂ the unobserved, respectively. Let the minimization factor in the observed random variable be φ_ε and ∂_ε , respectively. The separate factor between φ and ∂ parameters is given as:

$$C\partial = \varphi + \varphi_\varepsilon \quad (7-26)$$

$$\varphi = \partial + \partial_\varepsilon \quad (7-27)$$

Figure 7-19 shows the sum of weights of individual permissions in relation to their protection and threat level. To minimize error in the separation factor, we assumed that the minimization factor in both the observed and unobserved random parameters are distributed normally with their respective variances. Let the variances in the normal distribution for φ_ε and ∂_ε be $\sigma^2_{\varphi_\varepsilon}$ and $\sigma^2_{\partial_\varepsilon}$. The correlation that exists between φ and ∂ with their given variances is minimized and expressed as:

$$\rho = \frac{\sigma\varphi\partial}{\sqrt{\sigma^2\partial\sigma^2\varphi}} \quad (7-28)$$

Using Behseta Bayesian correlation principles [99], error minimization in ρ is expressed as follows:

$$\rho = \frac{\sigma\varphi\partial}{\sqrt{(\sigma^2_\varphi + \sigma^2_{\varphi_\varepsilon}) + (\sigma^2_\partial + \sigma^2_{\partial_\varepsilon})}} \quad (7-29)$$

However, if an error in the measurement is not correlated with the observed and unobserved random parameters, then the correlation of the unobserved parameter is greater than that of the observed random parameters accordingly.

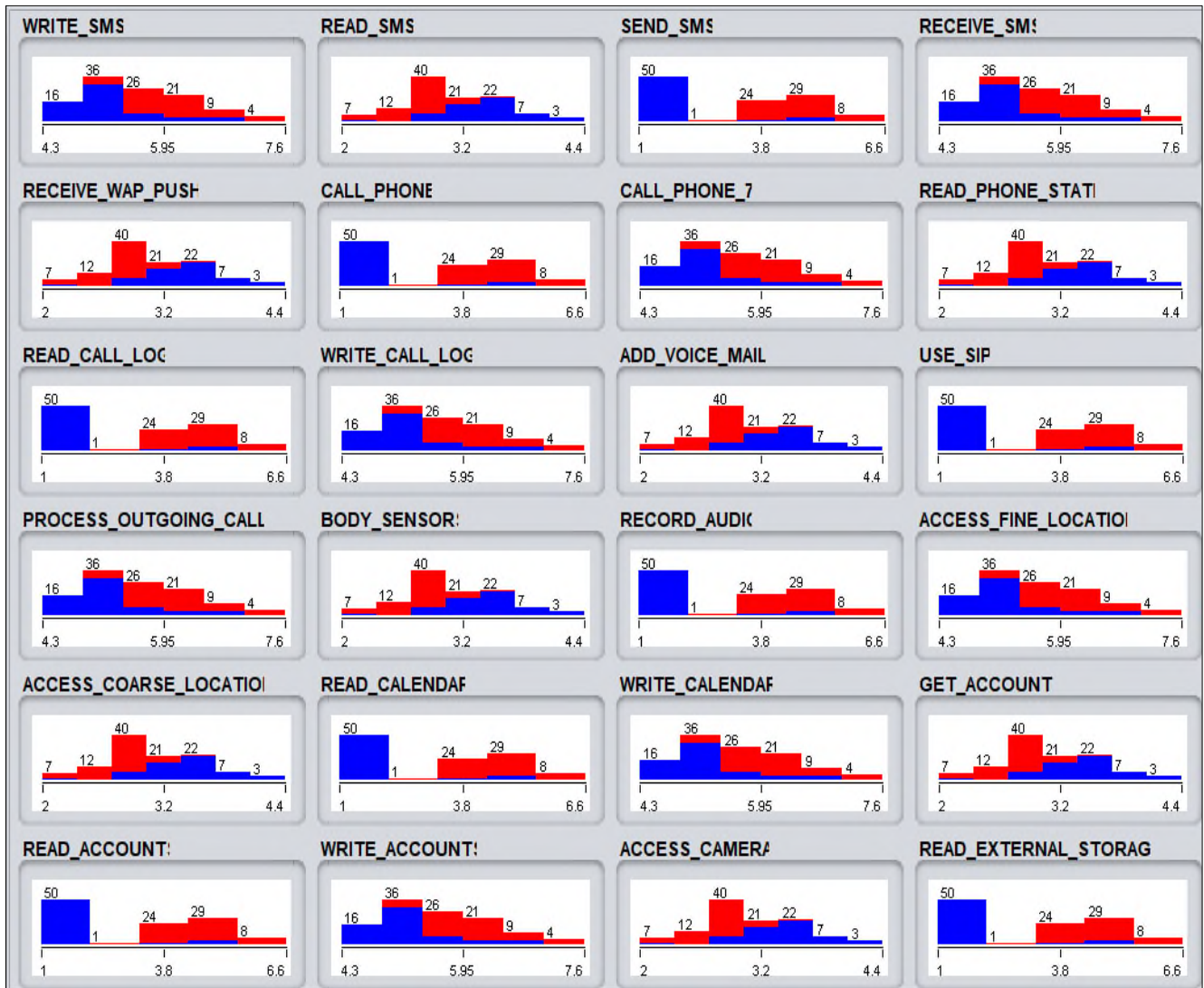
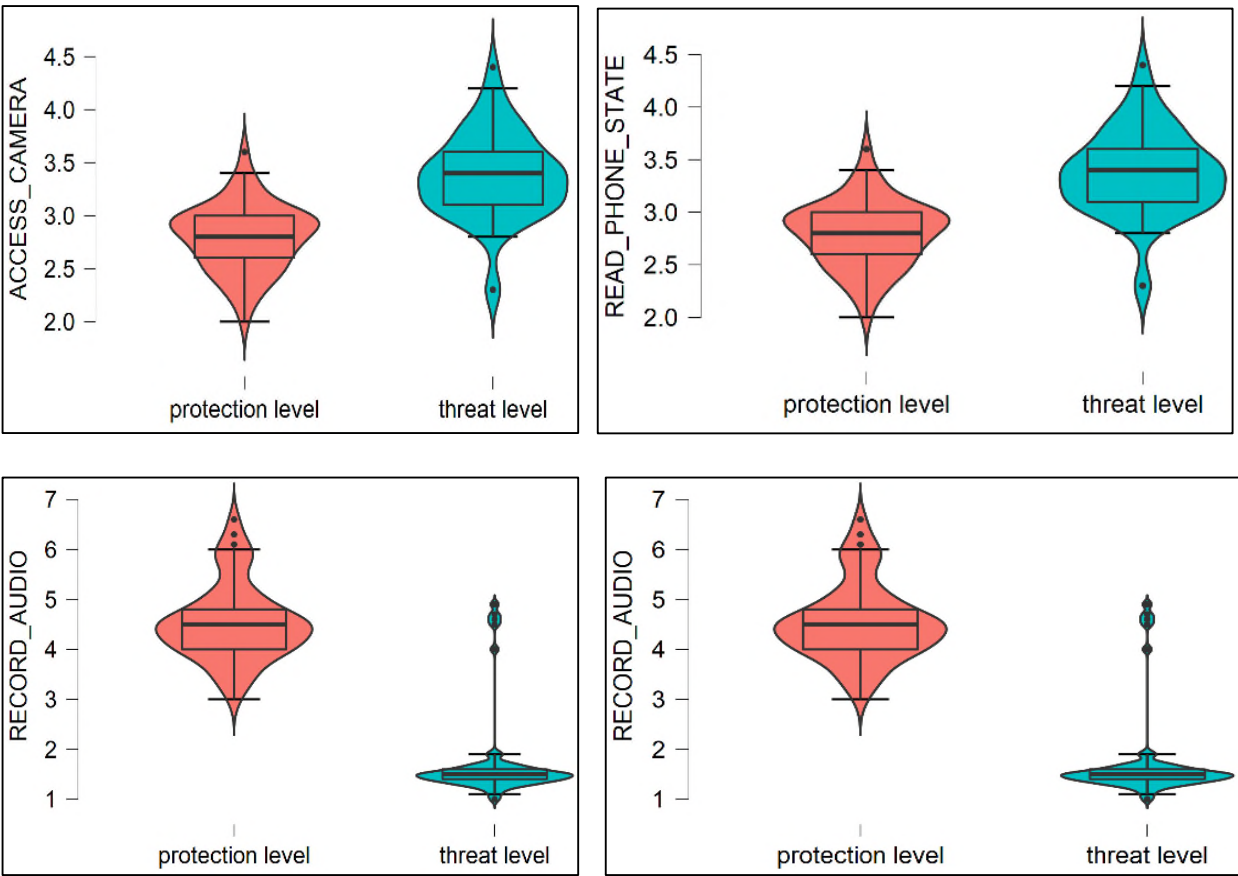


Figure 7-19 Exploratory factor plane visualization of permissions threat and protection level class correlation

7.5.1 Results and Discussion

This section of the paper discusses some of the results obtained from the model and their implications. The research identifies the relationship between the protection and the threat level permission could present. In presenting the result, we selected a few permissions to test their threat and protection level relationship based on our model formulation. Using the correlation tables, the rest of the permissions can be tested by any researcher, mobile security experts, and Android users

to determine the correlation that exists between the threat and protection level of each permission request. The Boxplots (figure 7-20) and the distribution plots (figure 7-21) for some permissions and their threat and protection levels relationship. We observed that the threat level is higher than the protection level in the ACCESS_CAMERA and READ_PHONE_STATE, respectively. On the other hand, the protection level in the RECORD_AUDIO and WRITE_EXTERNAL_STORAGE is higher than the threat level posed by those permission variables. Figure 7-22 shows the threat level of sensitive API extracted by our methodology.



Figures 7-20 Boxplots showing correlation between protection and threat level of some selected dangerous permission variables

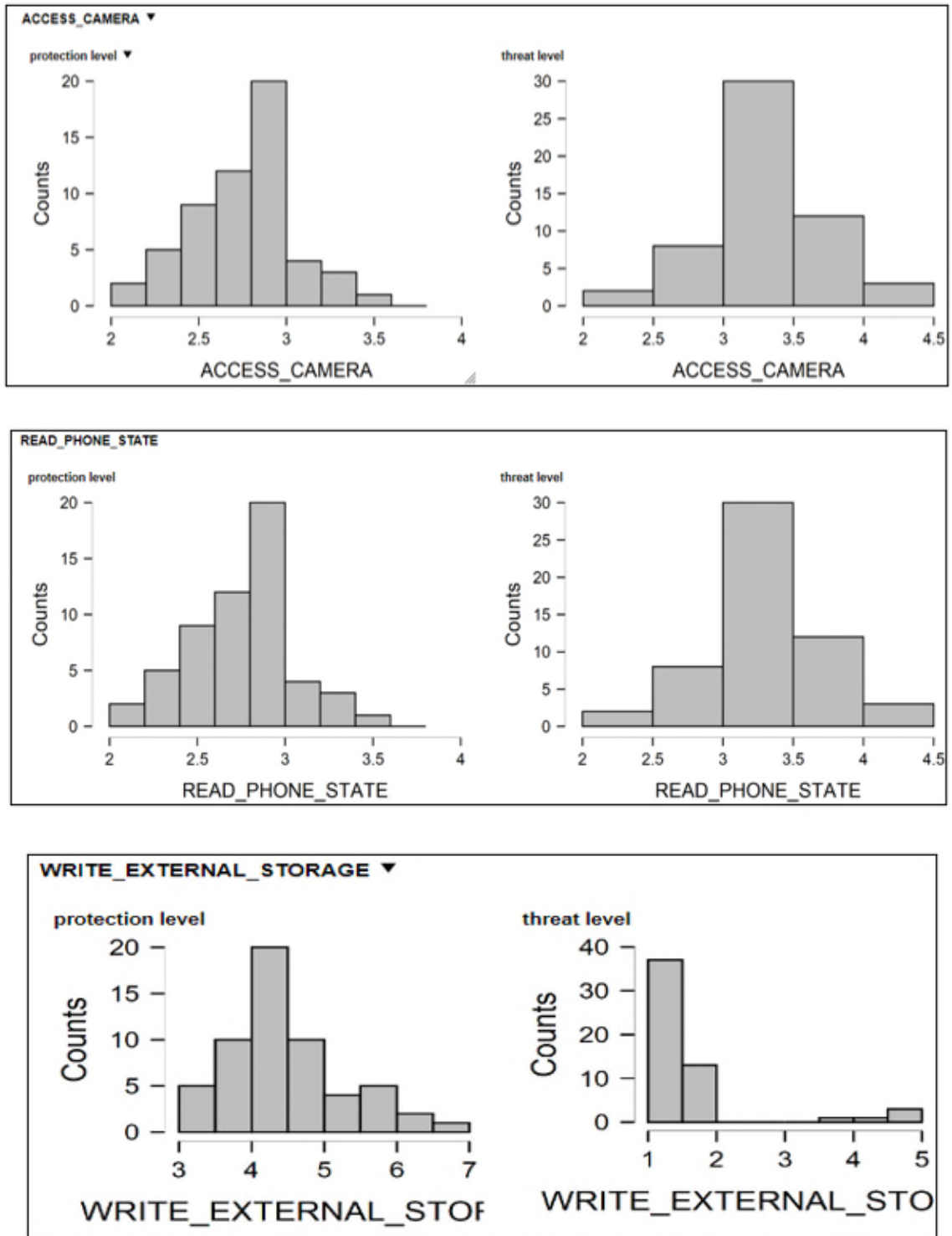


Figure 7-21 Scatter plots showing the protection and threat level of each permission per count

API: Landroid/content/pm/PackageManager;->getInstalledApplications Description: Gets applications installed on the device Caller Code: Lcom/tencent/mm/plugin/extqlauncher/d;->run()V Threat Level:  Path Index: 42
API: Landroid/content/pm/PackageManager;->getInstalledPackages Description: Gets packages installed on the device Caller Code: Lcom/tencent/mm/sdk/platformtools/bc;->Y(Landroid/content/Context;Ljava/lang/String;)Landroid/content/Intent Threat Level:  Path Index: 12
API: Landroid/hardware/Camera;->open Description: Open camera Caller Code: Lcom/tencent/mm/compatible/c/g;->mS()Lcom/tencent/mm/compatible/c/f; Threat Level:  Path Index: 12

Figure 7-22 Visualisation of some sensitive API showing their path index, threat level, caller code, and operation

We observed that each permission has a threat and protection level. Android permissions with the same protection level have the same threat level (see figure 7-23), especially if they are in the same. Permissions in the same clusters have similar densities and attributes. We observe that permissions that are in the same cluster may exhibit similar attributes and if coming from a malicious application, such permissions behave similarly in their effects. This implies that if any of them request access, the other ones in the same set will tend to execute some functionalities in the background.

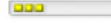
Permission Name	Protection Level	Threat Level
android.permission.ACCESS_COARSE_LOCATION	dangerous	
android.permission.ACCESS_FINE_LOCATION	dangerous	
android.permission.ACCESS_NETWORK_STATE	normal	
android.permission.ACCESS_WIFI_STATE	normal	
android.permission.AUTHENTICATE_ACCOUNTS	dangerous	
android.permission.ACCESS_WIFI_STATE	normal	
android.permission.AUTHENTICATE_ACCOUNTS	dangerous	
android.permission.BLUETOOTH	dangerous	
android.permission.BLUETOOTH_ADMIN	dangerous	
android.permission.BROADCAST_STICKY	normal	
android.permission.ACCESS_COARSE_LOCATION	dangerous	
android.permission.ACCESS_FINE_LOCATION	dangerous	

Figure 7-23 Classifying protection and threat level of certain permissions based on their usage

The research identifies that permissions are interposed in sets and can only be associated with a set with higher density. According to the results produced by the model, Bayes factors that are one-sided signify substantiation for the non-existence of correlation between threat level and protection level for both the inferred and observed relationship. However, this is not the situation in Android applications. As shown in figure 7-23, the result shows that Android malicious applications have different threat levels. Under the risk heading, the threat level posed by an application is ranked from 1 to 10 using rectangular dots. If the risk level of an application to a device is more than 6, our model classifies the permission as dangerous based on the protection level of the permission name. Conversely, if permission has a protection level whose threat level is less than 7, our method classifies such permission as dangerous.

From the result obtained, we can infer that if a malicious application has a low threat level, it is highly unlikely for such an application to dangerously infect a mobile device. At a moderate threat level, an attack on a device is potential but not likely. However, in substantial threat level applications, attack on mobile devices is likely and highly likely in severe threat levels (ranked up to 7 in our model). Permissions requested by applications with severe threat levels are likely to be dangerous permissions, as illustrated in figure 7-24. Applications with critical threat levels (ranked from 8 to 10 in our model) are highly likely to infect a mobile application and could request a combination of both normal and dangerous permissions.

Date	MD5	Package Name	Malware	Risk
2019-05-19 02:03:17	343586FE5E03215962A349A22A7EEF0A	vaytienmatnhanhonline.idong.vaytiennongngay	✓	*****
2019-05-19 01:53:37	5A9B119B168C9A0E7F32F14397656767	org.hola	✓	*****
2019-05-19 01:46:02	73861A335F4F477FFA24BD83D84DF534	orion.birthday.boyfriend	✓	*****
2019-05-19 01:36:24	6C0F7923C48EEEAD19702547D61E017C	nfo.webcam	✓	*****
2019-05-19 01:28:52	95B180F3EC12AEBAE34C83743FE80EC5	com.licketycut.hqhelper	✓	***
2019-05-19 01:28:52	B7FAFCF5F7527731F6AED39D1876F167	orion.birthday.brother	✓	*****
2019-05-18 11:24:59	4DE22C737A7A126D30022B2F356346D0	com.eyefilter.nightmode.bluelightfilter	✓	*****
2019-05-18 11:16:31	545BF57B736E929E4CBF233196247D4A	com.szyk.myheart	✓	****
2019-05-18 11:01:54	495A01F7ED4E6FFA08D7A85BBC7D045E	com.fifa.fifaapp.android	✓	***
2019-05-18 10:52:22	594B06C532625F715178A1FE0FDDA051	com.YovoGames.babycare	✓	****
2019-05-18 10:44:48	8D53160A5C477B3C63CDEDE008B59290	buttocksworkout.hipsworkouts.forwomen.legworkout	✓	*****
2019-05-18 10:30:38	9BE3784628ECEC1606B48C10375B3356	com.wBypassFRPLock_8137672	✓	*****
2019-05-28 07:02:32	39FD554E7BF39BCCFDEE6BC8829DDFBD	com.jamendoandoutly.mainpackage	✓	*****
2019-05-28 06:54:29	1A652905B726FAEEFC531904B2316102	com.estrongs.android.pop	✓	*****
2019-05-28 06:37:52	BFC132E014A4EBA1D9ED1BB33D17749B	com.JonsRonaldo.Wallpapers2018	✓	****
2019-05-28 06:26:26	80478AE8783B31064D79CA13BE60685C	com.android.vending	✓	*****
2019-05-21 21:07:23	1C85CF522580732626D9E5F0FA9E6F25	zozo.android.lostword	✓	*****
2019-05-21 18:42:17	511A12ABDE3D0A726C68F5A3A0D0E2E3	com.zing.mp3	✓	*****
2019-05-21 18:39:17	2A4C4FCAA9EAA0EAFB31D2D64634222D	com.ahmed.ali.kaitokid1234	✓	*****
2019-05-21 18:38:47	8760C49DF5F07AC23B8AD8AD79E7DEAE	com.dewmobile.kuaiya.play	✓	*****
2019-05-19 13:00:10	145C62B359B557B289D077CA1633ED16	com.opera.app.news	✓	*****
2019-05-19 12:59:40	D892380B96CA4303D436E9835CD3A47F	net.peakgames.OkeyPlus	✓	*****
2019-05-19 11:06:57	13D3F174F2CDECB0A04624A9A3F56AA	com.moonfrog.ludo.club	✓	*****

Figure 7-24 Android applications threat level obtained by our model

During clustering analysis, the results obtained (as shown in figure 7-25) demonstrated that most permissions in the same cluster have similarities in the feature distribution. For instance, READ_CALL_LOG and RECORD_AUDIO have some similarities in their structural distribution as can be seen in RECEIVE_SMS and WRITE_SMS respectively. This also demonstrated that permission in the same cluster has relevant features that are correlated.

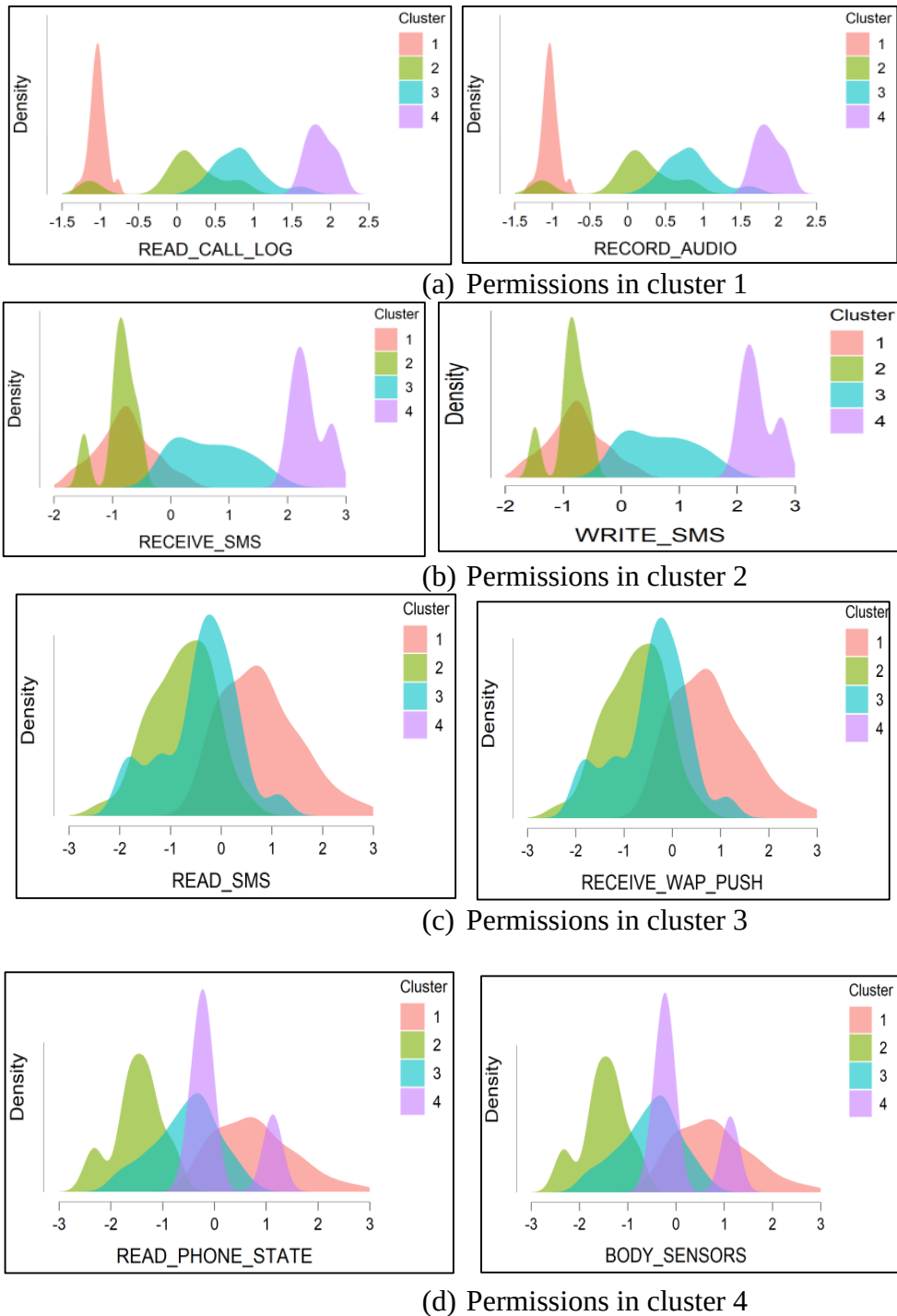


Figure 7-25 Cluster density plots showing dangerous permission-based clustering

The plot shows dangerous permission-based clustering where (a) Permissions in cluster 1. (b) Permissions in cluster 2. (c) Permissions in cluster 3. (d) Permissions in cluster 4. We observe that correlated permissions usually appear in locations that are the same. This suggests that Android applications that request CALL_LOG, SMS, or any permission sets that are correlated, similar permission sets will be requested underground and will be operational in the same region. Permissions in the same region execute frequent tasks such as read SMS and access audio and camera, among others. However, not all frequent permissions are dangerous. For example, a permission request to access the internet does not invoke internet access control obtained by an application. We then inferred that internet permission does not support adequately grained management or control of the device resources.

Comparatively, rather than use direct permission features to gain control of a device, malware authors may define some functionalities as access control. This aids in specifying how the malicious program can communicate with pre-defined applications to access the resources in question without necessarily requesting permission related to such resources. Based on the results obtained, we recommend improvement in the permission model of Android OS by differentiating internet permission access and other similar permissions into distinct groups and classes. Let a clear mechanism be provided for specifying Android permissions that are correlated with device resources. When the access control list is specified, this will help permissions definition by developers without using permissions attributes that are self-defined. For Android, numerous permissions have effects on Android end users and even developers.

It is then pertinent to configure a system to help understand each permissions' action concerning the device resources rather than over-requesting them by the developers simply because they want an app to function. This could be a good security strategy in preventing malware infection on mobile devices in general. However, studying actions associated with each permission is beyond the scope of our research.

Table 7-6 Correlation model comparison of some selected permissions

Models	P(M)	P(M data)	BF_M	BF₁₀	error %
RM Factor 1 + Correlation + SEND_SMS + RECEIVE_SMS + RM Factor 1 * Correlation	0.050	0.966	535.883	1.000	
RM Factor 1 + Correlation + RECEIVE_SMS + RM Factor 2 * Correlation	0.050	0.034	0.674	0.035	0.070
RM Factor 1 + Correlation + SEND_SMS + RM Factor 3 * Correlation	0.050	1.044×10 ⁻²¹	1.984 ×10 ⁻²⁰	1.081 ×10 ⁻²¹	0.537
RM Factor 2 + Correlation + RM Factor 1 * Correlation	0.050	1.005×10 ⁻²⁹	1.910 ×10 ⁻²⁸	1.041 ×10 ⁻²⁹	0.979
RM Factor 2 + SEND_SMS + RECEIVE_SMS + RM Factor 2 * Correlation	0.050	3.255×10 ⁻³⁸	6.184 ×10 ⁻³⁷	3.370 ×10 ⁻³⁸	0.411
RM Factor 2 + Correlation + SEND_SMS + RM Factor 3 * Correlation	0.050	1.184×10 ⁻³⁸	2.250 ×10 ⁻¹³	1.226 ×10 ⁻³⁸	0.321
RM Factor 3 + Correlation + RECEIVE_SMS + RM Factor 1 * Correlation	0.050	5.025×10 ⁻³⁹	9.547 ×10 ⁻³⁸	5.203 ×10 ⁻³⁹	0.960
RM Factor 3 + RECEIVE_SMS + RM Factor 2 * Correlation	0.050	1.378×10 ⁻⁴¹	2.617 ×10 ⁻⁴⁰	1.426 ×10 ⁻⁴¹	0.173
RM Factor 3 + Correlation + SEND_SMS + RM Factor 3 * Correlation	0.050	3.116×10 ⁻⁴⁹	5.919 ×10 ⁻⁴⁸	3.226 ×10 ⁻⁴⁹	0.265
Correlation + RECEIVE_SMS	0.050	1.790×10 ⁻¹³	3.402 ×10 ⁻³²	1.854 ×10 ⁻³³	0.007
SEND_SMS + RECEIVE_SMS	0.050	1.347×10 ⁻¹³	.559 ×10 ⁻³²	1.395 ×10 ⁻³³	0.580
Correlation + SEND_SMS	0.050	8.379×10 ⁻¹⁴	1.592 ×10 ⁻³²	8.676 ×10 ⁻³⁴	0.885
Correlation + SEND_SMS + RECEIVE_SMS	0.050	3.802×10 ⁻¹⁴	7.224 ×10 ⁻³³	3.937 ×10 ⁻³⁴	0.793
Null model (incl. subject)	0.050	7.182×10 ⁻¹³	1.365 ×10 ⁻³⁷	7.437 ×10 ⁻³⁵	0.529
SEND_SMS	0.050	5.468×10 ⁻¹³	1.039 ×10 ⁻³⁵	5.662 ×10 ⁻³⁵	0.740
Correlation	0.050	1.428×10 ⁻¹³	2.714 ×10 ⁻⁴¹	1.479 ×10 ⁻³⁵	0.660

7.5.2 Permission request and privileges escalation

This subsection provides a short description of how an attack leveraged misplaced trust in a permission request to escalate its privileges as illustrated in figure 7-26. Malware, like any other program, can potentially execute any permission from a standard user to root (administrator) based on the context it was originally executed within. However, an Android application version that has less permission is not restricted to access components of a more privileged application. For instance, consider X, Y, and Z represent three different Android applications, each running in its sandbox and having two components. In application Z, component Z_{z1} is protected by permission $p1$. Similarly, Z_{z2} by $p2$. Application Y is granted permission $p1$; hence, YZ_1 and YZ_2 can access Z_{z1} . However, Y is not protected by any permission, and its components are publicly accessible. X does not have any permissions, but its component Z_{x1} can access Z_{x1} . Z_{x1} is not accessible to Z_{z1} directly since it does not have permission $p1$; however, it can do so through component X_{y1} . This is a privilege escalation attack since the privileges of application X are escalated by the attacker to the privileges of application Z indirectly through Y, which indicates transitive usage of permission privileges. To prevent this kind of permission escalation attack, application Y must ensure that another application calling it must have permission $p1$.

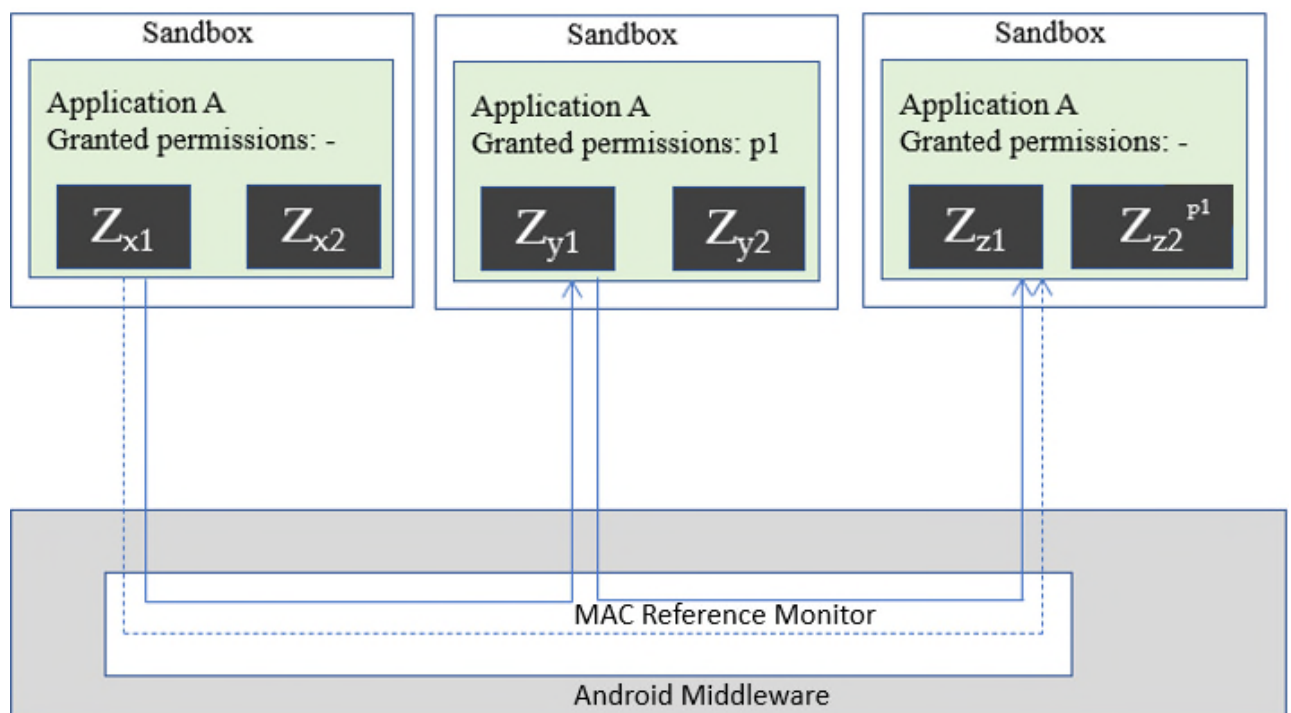


Figure 7-26. Demonstrating how an attack leveraged misplaced trust in a permission request to escalate its privileges.

7.6 Conclusion

In this work, we carried out factor analysis to determine the correlation between dangerous permission variables for Android malware samples using Bayesian correlation. The aim is to establish if there is a correlation in the set of a given permission request variables. This method is to assist in the analysis of the Android permissions security structure. Using SOM and t-SNE techniques, we visualized the Android malware data set by applying exploratory factor plane analysis, which reviewed patterns in which permissions are related. Large data set consisting of 12,267 malicious and 10,837 benign applications with different categories were used for the experiments.

Our results demonstrate our model can correctly recognize the correlation between dangerous permissions. We identified that permissions that are in the same protection level have the same threat level. Visualization results indicate that most permissions in the large subcategory were requested by very few Android applications, while frequently used permissions were those in the small subcategory. This signifies that there is no adequate expressiveness from permissions that are requested frequently. We infer that those not frequently requested could be disintegrated into the common class. As a result, we suggest adding finer granularity as a security approach for Android permissions that are frequently requested by applications will enhance this expressiveness and enhance Android security, especially when combined with the ones occasionally requested.

In general, this method will facilitate researchers infer correlation in the presence of estimation uncertainties in distributed clusters of Android permissions. The understanding of high threat permission will assist in enforcing security parameters at the permission security system of the Android platform and other mobile OS. These findings are significant for offering insights to assist mobile users in making low-risk decisions during application installations and when granting access to applications that have high threat levels, especially when similar permission names with the same protection level are requested. This provides insight to researchers and android users to infer the latent relationship between (1) Android permission requests from malicious Android applications; (2) the threat level and protection level of a permission request; and (3) to understand how a single grant of one permission may trigger background actions of others that are correlated with similar attributes. Although our model used three correlation coefficients, further research can be conducted on using more correlation coefficient parameters to model the relationship between permissions based on their threat and protection level. We used only the Android malware

data set for this model; we suggest that malware data set from other mobile platforms could be used as a further study to explore their permissions architecture using our methodology.

In conclusion, with the number of malicious applications increasing daily at a very fast rate, vulnerabilities are also increasing, making the platform for attackers wider. Thus, there is a need for a technique that provides a complete solution against permission privilege escalation attacks along with satisfying all the usability requirements. In the future, the impact on the security of Android users and vendors using Android permissions at different protection levels might be studied vigorously. Other permission systems on other mobile operating systems could be investigated, such as Windows and iOS. Moreover, further studies could investigate more if the defined permissions of the existing applications are fully utilized by them, or they are classified as over-privileged applications.

Acknowledgement

Funding: This research was funded by the Petroleum Technology Development Fund (PTDF), grant number PTDF/ED/PHD/AMA/1245/17/17.

REFERENCES

- [1] I. M. Almomani and A. Al Khayer, "A Comprehensive Analysis of the Android Permissions System," *IEEE Access*, vol. 8, pp. 216671–216688, 2020, doi: 10.1109/ACCESS.2020.3041432.
- [2] S. Peng, L. Cao, Y. Zhou, J. Xie, P. Yin and J. Mo, "Challenges and Trends of Android Malware Detection in the Era of Deep Learning," 2020 IEEE 8th International Conference on Smart City and Informatization (iSCI), Guangzhou, China, 31 Dec.-1 Jan. 2021, pp. 37-43, doi: 10.1109/iSCI50694.2020.00014.
- [3] Z. Ren, H. Wu, Q. Ning, I. Hussain, and B. Chen, "Ad Hoc Networks End-to-end malware detection for android IoT devices using deep learning," *Ad Hoc Networks*, vol. 101, p. 102098, 2020, doi: 10.1016/j.adhoc.2020.102098.
- [4] S. J. Hussain, "IMIAD : Intelligent Malware Identification for Android Platform," In *Proceedings of the 2019 International Conference on Computer and Information Sciences (ICCIS)*, Sakaka, Suadi Arabia, 3-4 April, 2019, pp. 1-6,
- [5] P. Faruki, "AndroSimilar : Robust Statistical Feature Signature for Android Malware Detection," In *Proceedings of the 6th International Conference on Security of Information and Networks*, 26–28 Nov. 2013; Turkey, pp. 152–159.
- [6] J. Feng, L. Shen, Z. Chen, Y. Wang and H. Li, "A Two-Layer Deep Learning Method for Android Malware Detection Using Network Traffic," in *IEEE Access*, vol. 8, pp. 125786-125796, 2020, doi: 10.1109/ACCESS.2020.3008081.
- [7] A. Malware, D. Based, and A. P. I. Pairing, "Android malware detection: a survey," *Scientia Sinica Informationis*, vol. 50, no.8, pp. 1148-77, 2020, doi: 10.1360/SSI-2019-0149.
- [8] M. Ashawa and S. Morris, "Host-Based Detection and Analysis of Android Malware: Implication for Privilege Exploitation," *Int. J. Inf. Secur. Res.*, vol. 9, no. 2, pp. 871–880, 2019, doi: 10.20533/ijisr.2042.4639.2019.0100.
- [9] M. Ashawa and S. Morris, "Analysis of Android Malware Detection Techniques: A Systematic Review," *Int. J. Cyber-Security Digit. Forensics*, vol. 8, no. 3, pp. 177–187, 2019, doi: 10.17781/p002605.
- [10] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android permissions demystified," *Proc. ACM Conf. Comput. Commun. Secur.*, pp. 627–636, 2011, doi: 10.1145/2046707.2046779.
- [11] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner, "Android permissions: User attention, comprehension, and behavior," *SOUPS 2012 - Proc. 8th Symp. Usable Priv. Secur.*, 2012, doi: 10.1145/2335356.2335360.
- [12] R. Kesler, M. E. Kummer, and P. Schulte, "Mobile Applications and Access to Private Data: The Supply Side of the Android Ecosystem," *SSRN Electron. J.*, no. 17, 2018, doi: 10.2139/ssrn.3106571.
- [13] L. Nguyen-vu, J. Ahn, and S. Jung, "Computers & Security Android Fragmentation in Malware Detection," *Comput. Secur.*, vol. 87, p. 101573, 2019, doi: 10.1016/j.cose.2019.101573.
- [14] A. Ifip, and D. Hutchison, "Data and Applications Security and Privacy," Springer: Cham, Switzerland, 2013.

- [15] X. Wei, L. Gomez, I. Neamtiu, and M. Faloutsos, "Permission Evolution in the Android Ecosystem". In Proceedings of the 28th Annual Computer Security Applications Conference (ACSAC '12), Association for Computing Machinery, New York, NY, USA, 2012, pp. 31–40, <https://doi.org/10.1145/2420950.2420956>.
- [16] S. A. Gorski and W. Enck, "ARF: Identifying re-delegation vulnerabilities in android system services." In Proceedings of the 12th Conference on Security and Privacy in Wireless and Mobile Networks (WiSec '19). Association for Computing Machinery, New York, NY, USA, May 2019, pp. 151–161. <https://doi.org/10.1145/3317549.3319725>.
- [17] F. Monrose, M. Dacier, G. Blanc, and J. Garcia-Alfaro, Research in Attacks, Intrusions, and Defenses: In Proceedings of the 19th International Symposium, RAID 2016, Paris, France, 19–21 Sept. 2016.
- [18] Y. Zhauniarovich and O. Gadyatskaya, "Small changes, big changes: An updated view on the android permission system," Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 9854 LNCS, pp. 346–367, 2016, doi: 10.1007/978-3-319-45719-2_16.
- [19] S. Malek, "EvoDroid: Segmented Evolutionary Testing of Android Apps," Springer: Cham, Switzerland, 2014; pp. 599–609, <https://doi.org/10.1145/2635868.2635896>.
- [20] D. Gallinger, S. Zanero, and V. N. Venkatakrishnan, "Practical Exploit Generation for Intent Message vulnerabilities for Android," . In Proceedings of the 5th ACM Conference on Data and Application Security and Privacy (CODASPY '15). Association for Computing Machinery, New York, NY, USA, March, 2015, 155–157. DOI:<https://doi.org/10.1145/2699026.2699132>.
- [21] K. O. Elish, D. D. Yao, B. G. Ryder, and V. Tech, "On the Need of Precise Inter-App ICC Classification for Detecting Android Malware Collusions." In Proceedings of IEEE Mobile Security Technologies (MoST), in Conjunction with the IEEE Symposium on Security and Privacy, 2015. Available online: <https://www.ieee-security.org/TC/SPW2015/MoST/papers/s2p4.pdf>.
- [22] A. Pekta and T. Acarman, "Deep learning for effective Android malware detection using API call graph embeddings," Soft Comput. pp. 1027–1043, 2019, doi: <https://doi.org/10.1007/s00500-019-03940-5>.
- [23] S. Cao, X. Sun, L. Bo, Y. Wei, and B. Li, "BGNN4VD : Constructing Bidirectional Graph Neural-Network for Vulnerability Detection," Inf. Softw. Technol., vol. 136, no. 5, pp. 106576, 2021, doi: 10.1016/j.infsof.2021.106576.
- [24] Y. Wu, J. Lu, and Y. Zhang, "Vulnerability Detection in C / C ++ Source Code With Graph Representation Learning," In Proceedings of the 2021 IEEE 11th Annual Computing and Communication Workshop and Conference (CCWC), Nevada, NV, USA, 27–30 January 2021; pp. 1519–1524.
- [25] H. Alasmary, A. Abusnaina, and R. Jang, "Soteria : Detecting Adversarial Examples in Control Flow Graph-based Malware Classifiers," In Proceedings of the 2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS), Singapore, 29 November–1 Dec. 2020, pp. 888–898, doi: <https://doi.org/10.1109/icdcs47774.2020.00089>.
- [26] Y. Feng, L. Chen, A. Zheng, C. Gao, and Z. Zheng, "AC-Net: Assessing the Consistency of Description and Permission in Android Apps," IEEE Access, vol. 7, pp. 57829–57842, 2019, doi: 10.1109/ACCESS.2019.2912210.
- [27] B. Rashidi, C. Fung, A. Nguyen, T. Vu, and E. Bertino, "Android user privacy preserving through crowdsourcing," IEEE Trans. Inf. Forensics Secur., vol. 13, no. 3, pp. 773–787, 2018, doi: 10.1109/TIFS.2017.2767019.

- [28] X. Wang, C. Li, and D. Song, "CrowdNet: Identifying Large-Scale Malicious Attacks over Android Kernel Structures," *IEEE Access*, vol. 8, pp. 15823–15837, 2020, doi: 10.1109/ACCESS.2020.2965954.
- [29] M. Irshad, H. M. Al-Khateeb, A. Mansour, M. Ashawa, and M. Hamisu, "Effective methods to detect metamorphic malware: A systematic review," *Int. J. Electron. Secur. Digit. Forensics*, vol. 10, no. 2, pp. 138–154, 2018, doi: 10.1504/IJESDF.2018.090948.
- [30] P. Mishra et al., "VMShield: Memory Introspection-based Malware Detection to Secure Cloud-based Services against Stealthy Attacks," *IEEE Trans. Ind. Informatics*, vol. 17, no. 8, pp. 1–9, 2021, doi: 10.1109/TII.2020.3048791.
- [31] A. Joux, "Attacks on stream ciphers," *Algorithmic Cryptanalysis*, Taylor and Francis Group, pp. 391–414, 2020, doi: 10.1201/9781420070033-17.
- [32] Y. Yuan, L. Wu, X. Zhang, and Y. Yang, "Side-channel collision attack based on multiple-bits." In *Proceedings of the 2017 11th IEEE International Conference on Anti-counterfeiting, Security, and Identification (ASID)*, Xiamen, China, 27–29 October 2017; pp. 1–5, <https://doi.org/10.1109/icasid.2017.8285732>.
- [33] L. Song et al., "AppIS: Protect android apps against runtime repackaging attacks." In *Proceedings of the 2017 IEEE 23rd International Conference on Parallel and Distributed Systems (ICPADS)*, 15–17 December 2017, Shenzhen, China, 2017, pp. 25–32, <https://doi.org/10.1109/icpads.2017.00015>.
- [34] D. Arnaldy and A. R. Perdana, "Implementation and Analysis of Penetration Techniques Using the Man-In-The-Middle Attack." In *Proceedings of the 2019 2nd International Conference of Computer and Informatics Engineering (IC2IE)*, 10–11 Sept. Banyuwangi, Indonesia, 2019; pp. 188–192, doi:<https://doi.org/10.1109/ic2ie47452.2019.8940872>.
- [35] Z. Aung and W. Zaw, "Permission-Based Android Malware Detection," *Int. J. Sci. Technol.*, vol. 2, no. 3, 2013.
- [36] S. Akleyek, "A novel permission-based Android malware detection system using feature selection based on linear regression," *Neural Comput. Appl.* vol. 1, no. 3, pp. 1–16, 2021, doi: 10.1007/s00521-021-05875-1.
- [37] J. T. McDonald, W. B. Glisson, N. Herron, and R. K. Benton, "Machine Learning-Based Android Malware Detection Using Manifest Permissions." In *Proceedings of the 54th Hawaii International Conference on System Sciences*, 2021, p. 6976, <https://doi.org/10.24251/hicss.2021.839>.
- [38] A. Mathur, L. M. Podila, K. Kulkarni, Q. Niyaz, and A. Y. Javaid, "NATICUSdroid: A malware detection framework for Android using native and custom permissions," *J. Inf. Secur. Appl.*, vol. 58, pp. 102696, 2021, doi: 10.1016/j.jisa.2020.102696.
- [39] H. Shahriar, M. Islam, and V. Clincy, "Android Malware Detection Using Permission Analysis," *Southeast Con. 2017*, pp. 1–6, <https://doi.org/10.1109/secon.2017.7925347>.
- [40] S. Al Ajrawi, A. Agrawal, H. Mangal, K. Putluri, B. Reid, and G. Hanna, "Materials Today : Proceedings Evaluating business Yelp ' s star ratings using sentiment analysis," *Mater. Today Proc.*, no. 80, 2021, doi: 10.1016/j.matpr.2020.12.137.
- [41] F. Paper, "Latent Semantic Indexing : An overview," *Technical report, INFOSYS*, pp. 1–16, 2000.
- [42] J. Li, L. Sun, and Q. Yan, "Significant Permission Identification for Machine-Learning-Based Android Malware Detection," *IEEE Trans. Ind. Inform.* vol. 14, no. 7, pp. 3216–3225, 2018, doi: <https://doi.org/10.1109/tii.2017.2789219>.
- [43] A. Arora, S. K. Peddoju, and M. Conti, "PermPair : Android Malware Detection Using Permission Pairs," *IEEE Trans. Inf. Forensics Secur.* vol. 15, pp. 1968–1982, 2020, doi: <https://doi.org/10.1109/tifs.2019.2950134>.

- [44] K. Khariwal and J. Singh, "IPDroid: Android Malware Detection using Intents and Permissions," In Proceedings of the 2020 Fourth World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4), London, UK, 27–28 July 2020; pp. 197–202, <https://doi.org/10.1109/worlds450073.2020.9210414>.
- [45] H. Wang, Y. Li, S. A. Khan, and Y. Luo, "Prediction of breast cancer distant recurrence using natural language processing and knowledge-guided convolutional neural network," *Artif. Intell. Med.*, vol. 110, no. 6, p. 101977, 2020, doi: 10.1016/j.artmed.2020.101977.
- [46] X. Wu, Y. Zhao, D. Radev, and A. Malhotra, "Identification of patients with carotid stenosis using natural language processing." *Eur. Radiol.*, vol. 30, pp. 4125–4133, 2020, doi: <https://doi.org/10.1007/s00330-020-06721-z>.
- [47] S. K. Peddoju and H. Upadhyay, "Natural Language Processing based Anomalous System Call Sequences Detection with Virtual Memory Introspection," *Int. J. Adv. Comput. Sci.* vol. 11, 2020, doi: 10.14569/IJACSA.2020.0110559.
- [48] Q. I. U. Xipeng, S. U. N. Tianxiang, X. U. Yige, S. Yunfan, D. A. I. Ning, and H. Xuanjing, "Pre-trained models for natural language processing: A survey. *Sci. China Technol. Sci.*, vol. 63, No.10, pp. 1–26, 2020, doi:10.1007/s11431-020-1647-3.
- [49] S. Jung, "AdMat: A CNN-on-Matrix Approach to Android Malware Detection and Classification," *IEEE Access* **2021**, vol. 9, pp. 39680–39694, 2021, doi: 10.1109/ACCESS.2021.3063748.
- [50] A. Mahindru and P. Singh, "Dynamic permissions based android malware detection using machine learning techniques," *ACM Int. Conf. Proceeding Ser.*, pp. 202–210, 2017, doi: 10.1145/3021460.3021485.
- [51] M. Leeds, M. Keffeler, and T. Atkison, "A comparison of features for android malware detection," *Proc. SouthEast Conf. ACMSE 2017*, pp. 63–68, 2017, doi: 10.1145/3077286.3077288.
- [52] L. Zhang and Q. Ji, "A Bayesian Network Model for Automatic and Interactive Image Segmentation," *IEEE Trans. Image Process.*, vol. 20, no. 9, pp. 2582–2593, 2011, doi: 10.1109/tip.2011.2121080.
- [53] J. Sullivan, A. Blake, M. Isard, and J. MacCormick, "Object localization by Bayesian correlation," *Proc. IEEE Int. Conf. Comput. Vis.*, vol. 2, no. 2, pp. 1068–1075, 1999, doi: 10.1109/iccv.1999.790391.
- [54] G. Dahman, J. Flordelis, and F. Tufvesson, "Cross-Correlation of Large-Scale Parameters in Multi-Link Systems: Analysis Using the Box-Cox Transformation," *IEEE Access*, vol. 6, pp. 13555–13564, 2018, doi: 10.1109/ACCESS.2018.2797418.
- [55] P. Sarder, W. Schierding, J. P. Cobb, and A. Nehorai, "Estimating sparse gene regulatory networks using a bayesian linear regression," *IEEE Trans. Nanobioscience*, vol. 9, no. 2, pp. 121–131, 2010, doi: 10.1109/TNB.2010.2043444.
- [56] M. Safayani, S. Momenzadeh, A. Mirzaei, and M. S. Razavi, "A latent variable model for two-dimensional canonical correlation analysis and the variational inference," *Soft Comput.*, vol. 24, no. 12, pp. 8737–8749, 2020, doi: 10.1007/s00500-020-04906-8.
- [57] H. Bai, N. Xie, X. Di, and Q. Ye, "FAMD: A Fast Multifeature Android Malware Detection Framework, Design, and Implementation," *IEEE Access*, vol. 8, pp. 194729–194740, 2020, doi: 10.1109/access.2020.3033026.
- [58] X. Zhang, A. Habibi Lashkari, and A. A. Ghorbani, "Classifying and clustering malicious advertisement uniform resource locators using deep learning," *Comput. Intell.*, vol. 37, no. 1, pp. 511–537, 2021, doi: 10.1111/coin.12422.
- [59] R. R. Ahirwal and M. Ahke, "Elliptic Curve Diffie-Hellman Key Exchange Algorithm for Securing Hypertext Information on Wide Area Network," *Int. J. Comput. Sci. Inf. Technol.*, vol. 4, no. 2, pp. 363–368, 2013.

- [60] D. Barrera, H. G. Kayacik, P. C. Van Oorschot, and A. Somayaji, "A methodology for empirical analysis of permission-based security models and its application to Android," *Proc. ACM Conf. Comput. Commun. Secur.*, no. 1, pp. 73–84, 2010, doi: 10.1145/1866307.1866317.
- [61] Y. Wang, F. Han, L. Zhu, O. Deussen, and B. Chen, "Line Graph or Scatter Plot? Automatic Selection of Methods for Visualizing Trends in Time Series," vol. 24, no. 2, pp. 1141–1154, 2018.
- [62] S. Schlegel, N. Korn, and G. Scheuermann, "On the Interpolation of Data with Normally Distributed Uncertainty for Visualization," *IEEE Trans. Vis. Comput. Graph.*, vol. 18, no. 12, pp. 2305–2314, 2012, doi:10.1109/tvcg.2012.249.
- [63] J. Lee, H. Son, G. Lee, J. Lee, S. Cho, and S. Lee, "Deep color transfer using histogram analogy," *Vis. Comput.*, vol. 36, no. 10, pp. 2129–2143, 2020, doi: 10.1007/s00371-020-01921-6.
- [64] L. Van Der Maaten, "Accelerating t-SNE using Tree-Based Algorithms," *J. Mach. Learn. Res.* vol. 15, pp. 1–21, 2014.
- [65] S. Arora, "An Analysis of the t-SNE Algorithm for Data Visualization." In *Proceedings of the Conference on Learning Theory*, vol. 75, pp. 1–8, 2018.
- [66] R. S. Faradonbeh, "Application of self-organizing map and fuzzy c-mean techniques for rockburst clustering in deep underground projects," *Neural Comput. Appl.*, vol. 32, no. 12, pp. 8545–8559, 2020, doi: 10.1007/s00521-019-04353-z.
- [67] N. A. Teanby, "An icosahedron-based method for even binning of globally distributed remote sensing data," *Comput. Geosci.*, vol. 32, no. 9, pp. 1442–1450, 2006, doi: 10.1016/j.cageo.2006.01.007.
- [68] Impact, "Contagio Malware Dump." Accessed on: Nov. 2, 2019. [Online]. Available: https://www.impactcybertrust.org/dataset_view?idDataset=1273.
- [69] VirusShare, "VirusShare.com - Because Sharing is Caring," Accessed on: Nov. 2, 2019. [Online]. Available: <https://virusshare.com/login>.
- [70] AndroZoo, "AndroZoo Access Conditions," Accessed on: Nov. 2, 2019. [Online]. Available: <https://androzoo.uni.lu/access>.
- [71] Impact cyber Trust, "The Drebin Dataset." Accessed on: Nov. 2, 2019. [Online]. Available: https://www.impactcybertrust.org/dataset_view?idDataset=1372.
- [72] M. Ashawa, "Android Permission Classifier : a deep learning algorithmic framework based on protection and threat levels," *International Journal of Security and Privacy*, vol. 4, no. 5, pp. 1–26, 2021, doi:10.1002/spy2.164
- [73] T. Van Erven and P. Harremoës, "Rényi Divergence and Kullback – Leibler Divergence," *IEEE Trans. Inform. Theory*, vol. 60, no. 7, pp. 3797–3820, 2014.
- [74] J. Aronsson et al., "The Barnes – Hut Hierarchical Center-of-Charge Approximation for Fast Capacitance Extraction in Multilayered Media," *IEEE Trans. Microw. Theory Tech.*, vol. 58, no. 5, pp. 1175–1188, 2010.
- [75] M. Khan and S. Shah Khan, "Data and Information Visualization Methods, and Interactive Mechanisms: A Survey," *Int. J. Comput. Appl.*, vol. 34, no. 1, pp. 975–8887, 2011.
- [76] E. Myasnikov, "Nearest Neighbor Search In Hyperspectral Data Using Binary Space Partitioning Trees." In *Proceedings of the 11th Workshop on Hyperspectral Imaging and Signal Processing: Evolution in Remote Sensing (WHISPERS)*, Amsterdam, the Netherlands, 24–26 March 2021; pp. 1–4, doi:10.1109/whispers52202.2021.9484041.
- [77] F. Nielsen, "Bregman vantage point trees for efficient nearest neighbor queries." In *Proceedings of the 2009 IEEE International Conference on Multimedia and Expo*, New York, NY, USA, 28 June–3 July, pp. 878–881, 2009.

- [78] T. Kohonen, "Essentials of the self-organizing map," *Neural Networks*, vol. 37, pp. 52–65, 2013, doi: 10.1016/j.neunet.2012.09.018.
- [79] M. Manera, "Exploratory factor analysis of rainbow trout serum chemistry variables," *Int. J. Environ. Res. Public Health*, vol. 18, no. 4, pp. 1–12, 2021, doi: 10.3390/ijerph18041537.
- [80] A. Vilorio, N. Bucci, M. Luna, J. P. Gutierrez, A. Parody, D. E. S. Bent, L. A. B. Lapoz, "Determination of dimensionality of the psychosocial risk assessment of internal, individual, double presence and external factors in work environments," *Springer: Cham, Switzerland*, vol. 10943, pp. 304–313, 2018.
- [81] D. Matzke et al., "Bayesian inference for correlations in the presence of measurement error and estimation uncertainty," *Collabra Psychol.*, vol. 3, no. 1, 2017, doi: 10.1525/collabra.78.
- [82] P. Tan and A. P. Filter, "An Adaptive Particle Filter Based on Posterior." In *Proceedings of the 2010 8th World Congress on Intelligent Control and Automation*, Jinan, China, 7–9 July 2010; pp. 5886–5890, 2010.
- [83] E. An and I. Ctor, "Modeling heavy-tailed bounded data by the trapezoidal beta distribution with applications," *Statistical journals*, vol. 1, no. 5, pp. 1-17, appear 2021.
- [84] J. Tonhauser and L. Matthewson, "Empirical Evidence In Research On Meaning." Master's Thesis, The Ohio State University and University of British Columbia, Ohio, OH, USA, pp. 200–215, 2015.
- [85] C. J. Vinkenburg and T. Weber, "Managerial career patterns : A review of the empirical evidence," *J. Vocat. Behav.*, vol. 80, no. 3, pp. 592–607, 2012, doi: 10.1016/j.jvb.2012.02.001.
- [86] N. Balakrishnan, E. Chimitova, N. Galanova, and M. Vedernikova, "Testing goodness of fit of parametric AFT and PH models with residuals," *Commun. Stat. Simul. Comput.*, vol. 42, no. 6, pp. 1352–1367, 2013, doi: 10.1080/03610918.2012.659824.
- [87] P. Wanke, Z. Chen, X. Zheng, and J. Antunes, "Sustainability efficiency and carbon inequality of the Chinese transportation system: A Robust Bayesian Stochastic Frontier Analysis," *J. Environ. Manage.*, vol. 77, no. 3, pp. 110163, 2020, doi: 10.1016/j.jenvman.2020.110163.
- [88] J. Zhang, B. Zhou, J. Guo, and X. Liu, "A Modified Bartlett Test for Heteroscedastic Two-way MANOVA," *J. Adv. Stat.* vol. 1, no. 2, pp. 94–108, 2016.
- [89] J. Niemi, "Parameter estimation: Normal model , unknown mean," *Lect. note. Iowa State University, USA*, pp. 1–35, 2015.
- [90] I. Parra-Frutos, "Testing homogeneity of variances with unequal sample sizes," *Comput. Stat.*, vol. 28, no. 3, pp. 1269–1297, 2013, doi: 10.1007/s00180-012-0353-x.
- [91] T. S. Sataeva and B. Y. Lemesko, "About properties and power of classical tests of homogeneity of variances," In *Proceedings of the 2016 11th International Forum on Strategic Technology (IFOST)*, Novosibirsk, Russia, 1–3 June 2016, pp. 350–354, 2017, doi: 10.1109/IFOST.2016.7884125.
- [92] M. Ghazal and A. Amer, "Homogeneity localization using particle filters with application to noise estimation," *IEEE Trans. Image Process.*, vol. 20, no. 7, pp. 1788–1796, 2011, doi: 10.1109/TIP.2010.2097272.
- [93] M. Lu, "An embedded method for gene identification problems involving unwanted data heterogeneity," *Hum. Genomics*, vol. 13, p. 45, 2019, doi: 10.1186/s40246-019-0228-0.
- [94] K. Kumar, *A Beginner's Guide to Structural Equation Modeling*, 3rd edn, Psychology Press: East Sussex, UK, vol. 175, no. 3. 2012.

- [95] R. Jacobs, P. Smith, and M. Goddard, "Centre for health and economics measuring performance: An examination of composite performance indicators." Centre of Health Economics, University of York: York, UK, 2004.
- [96] T. J. Hoskins, "Proofs of the Twin Primes and Goldbach Conjectures," pp. 1–33, 2019, Accessed on: Nov. 2, 2019. [Online]. Available: <https://arxiv.org/ftp/arxiv/papers/1901/1901.09668.pdf>.
- [97] C. P. Robert, N. Chopin, and J. Rousseau, "Harold Jeffreys's theory of probability revisited," *Stat. Sci.*, vol. 24, no. 2, pp. 141–172, 2009, doi: 10.1214/09-STS284.
- [98] C. Qin, J. Schlemper, J. Caballero, A. N. Price, J. V. Hajnal, and D. Rueckert, "Convolutional recurrent neural networks for dynamic MR image reconstruction," *IEEE Trans. Med. Imaging*, vol. 38, no. 1, pp. 280–290, 2019, doi: 10.1109/TMI.2018.2863670.
- [99] S. Behseta, T. Berdyeva, C. R. Olson, and R. E. Kass, "Bayesian correction for attenuation of correlation in multi-trial spike count data," *J. Neurophysiol.*, vol. 101, no. 4, pp. 2186–2193, 2009, doi: 10.1152/jn.90727.2008.

CHAPTER 8: GENERAL DISCUSSION

8.0 Introduction

The general discussion of this chapter is summarised in the context of the research question and objectives sets. Each objective is discussed in the context of its importance, key strength, limitations, and how they relate to future research. The chapter also provides the limitations and the thesis structure linked to the research objectives sets. Figure 8-1 summarised the structure of the thesis into three parts showing the thesis chapters' connectivity to research objectives. In the first part which are chapters 1 to 3, the thesis deals with the general review of malware advancement, evolution, and different infection strategies (Chapter 2). The analysis of malware detection techniques to identify their strengths and limitations based on the deployment approached was examined, as described in Chapter 3.

The second part of this thesis (Chapter 4) detailed building the malware lab for the research experiment. Chapter 5 is a preliminary study for statistical analysis of smart using LMD rules and framework development. Chapter 6 proposed a novel Android malware permission classification framework for mining and classifying Android permissions based on the characteristics of such applications in distinct contexts and user-defined operation circumstances. Chapter 7 (Paper 5) proposed a novel model for identifying the correlation between permissions requests.

8.1 Research question

To answer the research question, we examined the genre of malware using different aspects of their propagation modes (**Chapter 2 and 3** respectively) based on what has being studied by other people. we evaluated the impact of our techniques on identifying privileges that constitute the attack vectors (**Sections 5.6**) that smart malware uses to infect Android OS. To enhance malware classification as one of the strategies of preventing malware infection, we designed techniques that can identify privileges smart malware escalate (**Chapter 5**), a framework that classifiers Android permissions (**Chapter 6**), and a technique that detect malicious applications base on their permission threat and protection levels (**Chapter 7**). When combined, our robust techniques help to prevent malware infection on Android-based mobile platforms.

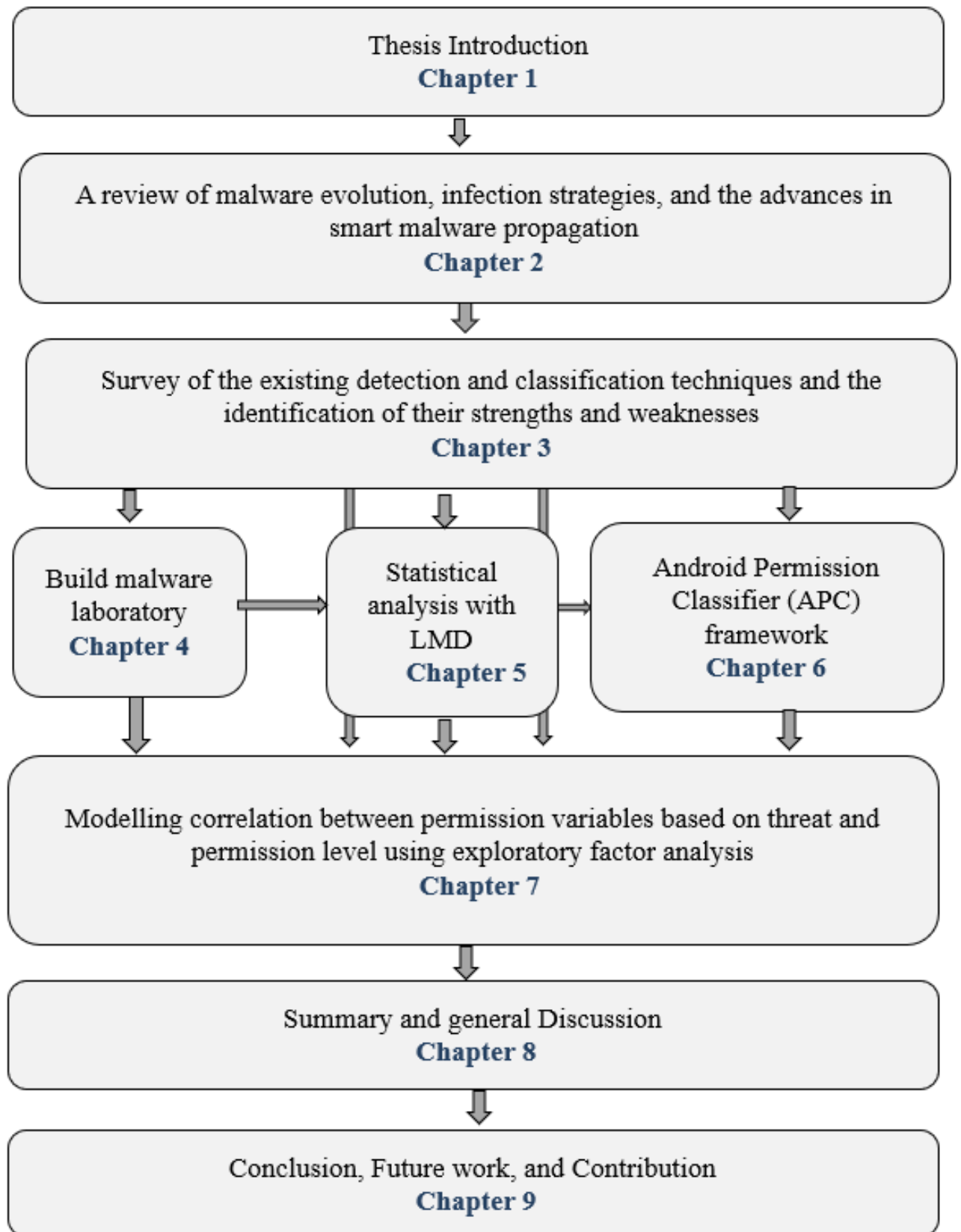


Figure 8-1: Research methodology demonstrating how the chapters are connected in this thesis

8.2 Research objectives

The research aims to develop methods based on the identification of attack vectors malware deploy to infect Android. We studied detection and classification techniques and tools to aid automated detection and analysis of sophisticated Android malware applications. Figure 9-1 summarised the structure of the thesis and the general discussion showing the thesis chapters' connectivity to research objectives. The general discussion of this chapter is summarised in the context of the research objectives sets.

8.2.1 Objective set 1

- *Conduct an up-to-date literature review on mobile malware's current state of the art*
- *Identify malware evolution trends and infection strategies*
- *Classify evolution trends based on their sophistication and attack vectors.*

To achieve this objective, we conducted a comprehensive study on mobile evolution and infection strategies looking at different related studies previously conducted in this area (**Chapter 2**). The objective aimed to advance understanding of the mobile malware history and its position in the cyber threat landscape by identifying existing gaps. During the survey, we identified major incidents of mobile malware infection strategies and their propagation approaches (**Table 2-1**). During the comprehensive review, we identified that evolution in writing malware brought a proportional complexity in its impacts on financial institutions, business, and information leakage (**Section 2.3.1**). Thus, malware moved from the realm of fun as did by script kiddies to financial gain, vengeance, system sabotage, cyberstalking, and political influence. Malware complexity makes detection difficult which requires having a tool to detect both traditional and evolving variants. This links to our **objective set 4**.

Analysing their evasion trend, we classified mobile malware evolution based on their behaviour (**Section 2.6**), privilege (**Section 2.6.1**), and attack goals (**Section 2.6.2 Table 2-4**) respectively. This classification is significant to identify the emergence of new malware variants with unique characteristics (**Table 2-3**) and the propagation strategy. For instance, in Category malware, based on the papers examined, our research referred to this group of mobile malware as modular malware. This group of mobile malware can gain administrative rights of a device and can perform a DDoS attack. One of the major significances of this objective is that we were able to identify

factors such as the lack of mobile OS standardization, lack of mobile OS cross-platform, lack of Bluetooth technological advancements in data communication, among others that limited the fast spread of the early mobile malware (**Section 2.5**).

In the systematic analysis, we also recognized factors that enhance the fast spread of recent mobile malware such as the advancement in threat dimensions, dark web advancement (**Section 2.3.1**), improved security and business communication on Tor using multi-signature transaction and encryption approach (**Section 2.3.2**), mobile platforms' integration with the IoT-based applications (**Section 2.4**), and the emergence of evasion techniques due to technological advancement (**Section 2.3.2.2**) among others. For the Android platform, open-source is one of the major factors that increase the rate of infection. We also identified infection strategies often used by mobile malware. While malware was written earlier, its rate of propagation was slow compared to recent malware. The speed of malware propagation is significantly enhanced by internet advancement. Internet connectivity is one of the major routes via which malware payloads are easily transmitted to other subsidiaries such as Bluetooth, SMS, cellular networks, Wi-Fi, physical access, USB-PC connections just to mention a few.

Understanding how malware is fast evolving by using different infection strategies we help to help develop a more robust and accurate detection tool. This will enhance the deployment of software solutions from enterprise mobility management to enhance the security of enterprise devices, observation of mobile application plugins and codecs during mobile application download and installation, and development of a mobile malware infection model to understand the immunity state of a mobile device during and after recovery from malware infection by application engineers. Though this objective was fully achieved, however, we did not cover in-depth malware evolution on PCs platforms. We identified factors that limited the fast spread of early malware and those that facilitate current malware propagation. The countermeasures provided as mitigation strategies in the chapter are for the Android platform. We did not generalize the risk mitigation strategies because we could not investigate other mobile platforms to compare their evolution and malware propagation and infection modes on those platforms. Therefore, generalisation may be risky.

8.2.2 Objective set 2

- *Examine existing malware detection techniques.*
- *Identify strengths and weaknesses of existing malware detection methods.*

There is an increased threat of malware, especially on mobile devices. To handle and prevent malware infections and propagation, different efforts are made by researchers to develop malware detection and classification methods. However, some of the detection tools are not sufficient in detecting the evolving and smart variants such as metamorphic malware. To have a scalable and accurate malware detection framework, there is a need to assess the existing techniques to identify their strengths and limitations. This objective set (**Chapter 3, Paper 2**) offered a systematic review of Android detection techniques identifying their unique strengths and limitations (**Table 3-1**).

By the findings, the signature-based detection approach generally implemented by anti-virus engines is very fast and efficient in detecting known malware variants whose signatures are known. However, the signature-based technique is not efficient in detecting variants with zero-day attributes just like the emulation-based technique (**Subsection 3.4.5**). Zero-day malware evades detection by signature-based techniques due to the use of evasion techniques such as obfuscation. Detection techniques that could not identify zero-day variants could not be able to identify malware that escalates privileges.

Other techniques such as permission and heuristic perform better than the signature approach by identifying variants that use network traffic for propagation (**Subsection 3.4.4**). However, a significant amount of time is required for scanning applications. False-positive and the inability to detect variants that do not generate traffic to sniff data constitute limitations of these techniques. We however did not investigate in detail their other weaknesses, but we believe memory analysis will be more promising for overcoming these limitations. This method requires a high level of expertise is needed for memory forensic analysis of malware.

Investigating the hybrid approach (**Subsection 3.4.3**), the synergy of static and dynamic methods produces better malware detection accuracy. By implication, feature interference of malicious applications during detection can be avoided when implementing a technique using this method. However, this might not be able to generate the malware sequence and call graph distribution state for detecting metamorphic variants that mutate their malicious code every time they execute. We observed that the dynamic approaches investigated (**Subsection 3.4.1**) virtualisation engine is

significant in preventing infection of the physical machines. This technique can detect malware variants that steal sensitive information because of its ability to identify function parameters, instruction inflow, and system behaviour. However, the identification of the vulnerability code in sophisticated variants that conceal their existence may prove difficult resulting in biases during feature selection.

8.2.3 Objective set 3

- *Identify privileges that malware exploits to affect Android mobile devices.*

Android provides regular updates to enhance security and functionality. Based on the architecture of the Android system, the updates constantly revamp entirely the running process and add several files to the Android architecture having applications and sensitive data of users. These updates are often requested while their security implications are most at times not clearly labelled for users. To achieve this objective set (**Chapter 5, Paper 3**), we carried out a study to detect Android malicious applications focusing on identifying privileges Android malware escalate. We centred our evaluation (**Section 5.6**) of the escalated privileges on the major activity parameters including permissions requested (**Subsection 5.6.1**), network communication (**Subsection 5.6.2**), payload, and location monitoring (**Subsection 5.6.3**), battery depletion (**Subsection 5.6.4**), other privileges that malware exploit.

This result shade illumination on critical security susceptibilities which privilege escalation using permissions one of those frequently used to explore pileup weaknesses in Android. As demonstrated (**Subsection 5.6.1**), some malicious applications can strategically use the permissions request approach to declare some set of privileges. Privileges declared by malware may not be functional in that very version of Android but wait on for an update to occur before becoming functional on the upgraded version. This implies that when permission and other vulnerabilities are exploited by a malicious application by declaring some set of privileges, the protection levels (**Sections 7.4**) settings are determined by the malicious application. In addition, other vulnerabilities such as the pileup can be acquired via signature permissions which are further used to execute other payloads (**Subsections 5.6.3**) including file access and contamination (**Subsections 5.6.1, Figure 5-7**), stealing sensitive URLs connected data (**Subsections 5.6.1, Figure 5-6b**), among others. According to Xing et al. [21], when critical Android security privileges are exploited, it could lead to alterations in the device security configurations. Thus,

obstructing critical services and patches installations on the device could have prevented other exploit chances hackers can leverage to attack the device. Though we have identified in this objective set those privileges which serve as attack vectors that malware exploit

Knowledge of privileges escalation using vulnerabilities such as pileup vulnerability is significant to support Android manufacturers and users take strategic approaches conservatively to prevent substituting Android resources from anonymous origins during an update. Our research has developed a framework to classify Android permissions (**Chapter 6, Paper 4**) and to classify permissions and applications threat levels as a mitigation strategy against this threat. However, this framework is not tested on other data and does not focus on mitigating pileup susceptibilities. This opened for future research on enhancing Android security by mitigating pileup vulnerabilities malicious applications exploit to escalate privileges.

8.2.4 Objective set 4

- *Develop a framework for identifying privacy preferences such as permission features emerging, and smart malware use as attack vectors on Android.*

This objective set requires designing a framework to classify Android permissions based on protection and threat levels. The increase in the number of applications and the evasion techniques requires having a more robust system to prevent malware infection. We identified that some of the existing techniques (**Chapter 3 Section 3.4**) have significant limitations (**Section 3.6, Table 3-1**) that require improvement. To improve this requires having effective an automated model that uses a different features detection approach. This led us to propose a novel Android malware permission classification framework (**Chapter 6**) for mining and classifying Android permissions based on the characteristics of such applications in distinct contexts and user-defined operation circumstances. In our study, we focused on permission classification based on the threat level of such permission requests, and the corresponding protection level was achieved by our model. We developed an Android permission classifier that extracts application feature similarities (**Subsections 6.3.2.1, 6.3.2.2, 6.3.2.3, and 6.3.2.4**) that combines the feature vectors to form a vectors database (**Section 6.4, Algorithm 1**). The structure of the database was composed of different components where all the features were pruned and classified and classified.

We identified that Pruning based on these levels has limitations since not all permissions are used. Also, support-based ranking of permission approach during the pruning, low requested permissions may not have a significant influence on malware detection performance. For example, permissions such as `BIND_TEXT_SERVICES` (**Table 6-7**) have low support are mostly used by benign applications. As a result, we considered that `BIND_TEXT_SERVICES` are requested by benign applications. Therefore, depending only on the negative rate and support-based approaches is not sufficient in accurately determining permissions that have high support for ranked attributes when determining their correlation matrix, eigenvalue, and eigenvectors. This forms one of the weaknesses in our framework. The associated rules (**Table 6-4**) for the mining process on the instances and attributes of the dataset for a depth-first search helps in finding associative patterns in the dataset. Building a classifier using a decision tree from data having many attributes involves considerable pre-processing exercise. As a result, pruning the attributes using a decision table (see **Table 6-4**) is one of the major strengths of our framework.

The feature vectors of the metrics were the same as the feature vectors in the new Android applications versions while feature vectors for old applications were different from the metrics. This helps our framework in detecting update attacks smart malware use to infect Android devices. For instance, if a benign application is repackaged with malicious code and made available in the new version of Android or any mobile application, the variation in metrics values will be analogous to that of the value of the original metric present in the malicious application. This is significant because when a legitimate application has already been installed by a user request for an update, the application does not ask the user for the second time for permissions again. An update attack occurs when an updated part of a benign application is embedded with malicious code. Identifying similarities and differences in the permissions metrics values during applications update is significant in detecting update attacks by smart malware. This is one of the major strengths of our framework. The APC framework is robust in automatically creating patterns of permission requests to accelerate applications classifications. The results demonstrated that automated classification of permissions using the regularisation approach (**Section 6.7.2**) increases feature diversity and minimises search space complexity in the performance metric (**Section 6.8.2, Figure 6.4**).

The results of the experiments presented show a higher AUC achievement of 95% (**Figure 6-3**) compared with the state-of-the-art permission request classification and other machine learning classifiers (**Table 6-9**). We suggested a regularization function for feature diversity to reduce filtering. This helps to increase feature diversity selection and generalization of the framework improvement. Based on the results obtained, we deduce that the classifier's performance and accuracy are a function of the features extracted and the data set distribution. This is because distinct feature representation techniques are used for distinct data sets in their categorisation approach. Thus, we suggest having sufficient training data set samples whose distribution is unbiased in both classes to increase accuracy and reduce FPR. Though the technique can be generalised for detecting malware to all mobile-based applications, it does not include Android applications dynamic tracing and has not to be tested using data set from other mobile platforms. Android has the same file format which is APK. However, feature extraction experiments in chapter 6 revealed that the APK structure varies due to a mixture of different benign and malicious applications from different versions of Android. Studying each of the application's APK structures in relation to their permissions declaration was challenging and time-consuming. I decided to prune all the permission features based on the versions of the applications. The limitation of this approach is that not all permissions might be used by sophisticated malware.

8.2.5 Objective set 5

- *Identify the relationship between permissions requests.*
- *Identify the relationship between threat and protection level to facilitate sophisticated malware detection.*

This objective requires identification of correlation between Android permissions sophisticated malware deploy to affect Android mobile devices through the model design method (**Sections 7.3**) using t-Distribution stochastic neighbor embedding (**Section 7.3.2**), the Self-Organisation Map (**Section 7.3.3**), and non-parametric functions [**Sections 7.3.5.1, 7.3.5.2, and 7.3.5.3**]. From the exploratory analysis, we recognised that every permission depicts a supplementary factor or dimension inside which Android applications are formulated and expressed. We observe that two or more permissions are related, provided there is a similarity existing in their factor plane during visualization. We observed that each permission has a threat and protection level and Android permissions with the same protection level have the same threat level (**Section 7.4, Figure 7-23**).

The factor analysis visualised the related permissions by providing factor distribution of each permission in specific dimensions. For instance, some pairs of permissions like read and write contacts and call logs are similar in their characteristics. We suppose such call log permissions were separated for developers for general call log services without knowing the call details of the user. For instance, correlated permissions such as access fine and coarse locations are designed to provide a broad geographical region without necessarily recognizing the precise coordinates of your GPS. We observe that correlated permissions usually appear in locations that are the same. This suggests that Android applications that request CALL_LOG, SMS, or any permission sets that are correlated, similar permission sets will be requested underground and will be operational in the same area.

8.3 Research general evaluation

We observe that correlated permissions usually appear in locations that are the same especially traditional permissions frequently requested. This suggests that Android applications that request CALL_LOG, SMS, or any permission sets that are correlated, similar permission sets will be requested underground and will be operational in the same region (**Subsection 7.4.4 Figure 7.9**). Permissions in the same region execute frequent tasks such as read SMS, access audio and camera, among others. However, not all frequent permissions are dangerous. For example, a permission request to access the internet does not invoke internet access control obtained by an application. We then inferred that internet permission does not support adequately grained management or control of the device resources.

Comparatively, rather than use direct permission features to gain control of a device, malware authors may define some functionalities as access control. Thus, specifying how the malicious program can communicate with pre-defined applications to access the resources in question without necessarily requesting permission related to such resources especially if they have similar cluster attributes (**Section 7.5.1**). Based on the results obtained, we recommend improvement in the permission model of Android OS by differentiating internet permission access and other similar permissions into distinct groups and classes. Let a clear mechanism be provided for specifying Android permissions that are correlated with device resources. When the access control list is specified, this will help permissions definition by developers without using permissions attributes that are self-defined. For Android, numerous permissions have effects on Android end users and even developers. It is then pertinent to configure a system to help understand each permissions'

action concerning the device resources rather than over-requesting them by the developers simply because they want an app to function. This could be a good security strategy in preventing malware infection on mobile devices in general. However, studying actions associated with each permission is beyond the scope of our research.

As discussed in **Section 7.9**, these results are substantial to understanding how a grant of one permission could trigger the installation of others that are correlated with similar attributes of some self-defined permissions. Our approach in this thesis is on Android based-permissions model which we believe the technique is appropriate to investigate permission usage and handling shifts in Android, but this can be extended to other mobile platforms which use manifest for accessing services and resources. It is essential to mention that Android applications used for our research have correlated permissions (**Section 7.9**), this in no way suggests that both permissions were requested. Instead, it implies that such applications have comparable attributes when requesting read or write account functionalities. This may cause a false-positive rate during malware classification if the classifier is solely based on the exploratory factor analysis only.

We observed that visualising sensitive API using exploratory factor plane analysis (**Section 7.3.4**), and have identified that permissions in the same region have the same protection level. This implies that if two or more permission names have the same protection level, their threat level will certainly be the same (**Section 7.9, Figure 7.23**). For example, access fine and coarse location both have the same protection and threat level in their usage. However, this is not the situation with Android applications. As illustrated (**Figure 7.24**), application packages have different threat levels. This result implies permissions with the same threat level attributes may perform similar functionalities.

This result is similar to the one obtained by [18] which states that “applications that have both the send and write or the receive and write SMS permission labels are ‘dangerous’ due to being able to intercept or transmit messages without the user’s knowledge”. Though correlation exists between permissions, this doesn’t mean that both permissions are requested at the same time or during installation. Rather, it implies that applications have similar characteristics when similar functionality is requested. The implication of this being that this may run into false positive classification when working on malicious applications. For instance, two malicious applications namely com.szyK.myheart and com.YovoGames.boyfriend (**Figure 7-24**) with equal threat levels may exhibit more than one functionality in their operation. Understanding and classifying the threat level of applications can be used to study and classify mobile malware behaviour and their

propagation routines without necessarily understanding the application's functions based on trust. This can be applied to other permission-based systems such as Google Chrome operating system extension which deploy Android-like permissions when gaining access to the enhanced trust.

There are many various methods and approaches for modelling and analysing the relationship between variables in a large and complex data set some of which include scattering plots, interpolation, histogram, among others. However, I chose to combine t-SNE (**Section 7.3.2**) and SOM (**Section 7.3.3**) for visualising permissions correlation because their combination gives a nearly accurate, reproducible, and very clear analysis of pairwise similarities. Because of the time constraint, I did not explore in depth the restrictions of using other traditional visualisation techniques apart from our knowledge of the fact that the listed known techniques are limited in analysing data sets with high dimensions and complexity like malware.

Finally, because of the time constraint, we did not investigate in-depth actions associated with each pair of correlated permissions. The permission classification framework and the permission correlation model in **chapters 6 and 7** show how robust and diverse our techniques are in detecting and preventing malware infection. However, because of the time limitation, we could not test the techniques on other data set. This implies that these techniques may not be appropriate for generalisation using data set from other mobile platforms.

8.4 Summary

Attack by malware on Android devices is dangerous due to the number of impacts such as financial loss, sensitive data leakage, and control of the infected device by the malware. Detection tools already in existence need improvement to detect malware variants that are smart and sophisticated due to some of their limitations in identifying smart malware due to their ever evasion strategies malware writers deploy. Consequently, the existing techniques are less effective in preventing smart malware infection. As a result, there is a need for an automated robust method that can identify attack vectors often used by sophisticated variants to attack Android. To mitigate malware threats on Android, we developed novel techniques that automatically identify privileges that malware escalates, automatically detects and classifies applications and their permissions into different categories depending on their risk levels. The results of our research show permission as one of the attack vectors and privileges mostly exploited by smart malware to infect Android and further exploit other vulnerabilities such as pileup.

CHAPTER 9: FUTURE WORK, CONTRIBUTION, AND CONCLUSION

9.0 Introduction

Smartphone increase and their computational advancements are related to the increase in the number of fake and malicious applications targeting smartphone platforms. Among the mobile OS, Android is the leading OS and is still rapidly increasing in popularity due to factors such as open OS, cheap to purchase, among others (**Chapter 1**). The open-source nature of Android and its popularity has increased the number of malware targeting the Android OS platform. While their detection techniques such as the use of anti-virus scan engines, hackers keep developing different evasion techniques to avoid been detected. As a result, there is a need to develop a framework with high feature diversity to detect evolving malware.

In addition, cybercriminals keep deploying different strategies to infect mobile devices with malware because of the financial profits they benefit from. Therefore, it is pertinent to keep developing different techniques to enhance Android security. A thorough review of malware and Android security state of the art was presented in this thesis (**Chapter 2**). This research addresses some open issues associated with smart malware automation detection through the use of permission features and their correlations in detecting Android malware. Other open issues identified are discussed in this chapter. The conclusion part of this thesis is provided in this chapter. The conclusion is presented in two sections. In the first section, we discuss some of the open issues identified during the research which are offered for future work. In the second section, the major contributions and how such contributions fulfil the set objectives are also summarised.

9.1 Other open issues for future research

Malware attacks on smartphones are still a global challenge with many open issues that need to be addressed using new solutions. The following open issues directly relate to our results and weaknesses of our work that need to be addressed and expanded to cover greater areas. Some of the open issues are the unknowns within our research that need further investigation.

9.1.1 Enhancing the security of Android by adding finer granularity

In Chapter 7, we proposed a model that demonstrated how risky some malicious applications are based on their threat level (**Section 7.4**). Visualisation results (**Section 7.4, Figure 7-18 and 7-19**) indicate that most permissions in the large subcategory were requested by very few Android applications while frequently used permissions were those in the small subcategory. This signifies that there is no adequate expressiveness from permissions that are requested frequently. The importance of this contribution will assist in enforcing security parameters at the permission security system of the Android platform and other mobile OS. These findings are significant for offering insights to assist mobile users to make low-risk decisions during application installations and when granting access to applications that have high threat levels especially when similar permission names with the same protection level are requested (**Chapter 7, Section 7.4**).

Our model can be expanded to other mobile permission-based platforms such as Windows and Symbian. We infer those permissions not frequently requested could be disintegrated into the common class. As a result, we suggest further research on adding finer granularity as a security approach for Android permissions that are frequently requested by applications. I hope adding finer granularity will enhance this expressiveness and enhance Android security, especially when combined with the ones occasionally requested. However, we are not sure if finer granularity will increase the model complexity.

9.1.2 Trusted applications

Trust is one of the major issues when determining the legitimacy of an application due to certain factors such as identification of the application developer's signature and security surrounding its safety. Trust of whether the applications were adequately checked before being uploaded in the marketplace for download is a big challenge. When the signature of the developer of an application is identified, the application's integrity in a way is ascertained. Though, this may not be adequate since there are thousands of applications that are submitted daily [1] by third-party operators and the majority of users pay no attention during applications' download from third-party markets.

The evasion techniques such as repackaging have made application trust a major issue as discussed (**Chapter 2 Section 2.1**). As a result, further research should be carried out to enhance application trusts, especially in the application store. Further research in software integrity is significant as the

number of applications has been developed and uploaded daily on market stores. Android permission classifier framework [2] developed could offer a variety of benefits such as identifying malicious permission attributes. However, the framework is not a tool for testing the application's integrity and trust-building in software.

The study of [3] presents trust computation in Mobile networks and how to manage this issue of software integrity from alternative markets which is not very sufficient in malware prevention. In our results (**Section 5.6.2**), we demonstrated how to use the invoke method `android.conn.CONNECTIVITY_CHANGE` to trace network communication to monitor data leakage of an application. Our results are important to aid mobile organisations to examine the relationship between the contacted domains and the contacted IP address. Thus, providing a hint of the countries with which the malware interacted. This is significant for digital forensic investigation during malware outbreaks. However, our research is limited to social networks interactions and cannot help in applications that are not social work based. To enhance trust on Android devices, the research of Tobias et al. [4] proposed an attestation technique as a security mechanism to check and measure applications integrity, especially on Android platforms. However, this technique needs procedure and semantic validation. It is therefore essential to build new schemes that could guarantee application integrity and authentication even on rooted Android devices.

9.1.3 Malware attacks on RFID technology-based applications

Apart from smartphones, other smart devices and systems using Radio Frequency Identification have security issues that need solutions. For example, RFID-based applications such as e-passport, race timing, and e-health applications are associated with malware and other security threats. Malware threats on RFID applications and other smart devices such as smartwatches, smart TV, smart fridges, among others are often underrated. These RFID applications and other smart devices are connected to smartphones and are supported internet-wise. Due to the connectivity of these devices during operation, malware attacks on them could in a way infect mobile devices because of the file exchange and communication link between them. Most of these RFID applications employ cyberspace-empowered mobile devices which store data that is attractive to criminals for attacks. For example, medical devices such as implantable medical devices IMDs [5] which hold

valuable and sensitive information may be a highly alluring target for ransomware for financial significance as discussed in the result (**Chapter 5, Section 5.6.1**).

There are related works that focused on preventing IMDs failure and sensors communication to control patients' records during and after aggregation [6]. Though this approach is based on the security and privacy of these devices, securing only the communication protocol without giving attention to smartphone-connection privacy cannot solve malware attack tension. For example, the identity of an IMD device connected to smartphones can be revealed to hackers when attacked. If the attack is ransomware, it might threaten to broadcast the health situation of the affected patients which could lead to social stigma. In addition, an adversary may have unauthorized access to determine remotely a patient's number of IMDs when smartphone-IMDs existence privacy is compromised which could introduce malicious communication with smartphones, external equipment, and IMDs. Apart from IMDs, other communication technologies embedded with NFC functionalities that are associated with mobile devices pose security threats. These devices can reveal the location and other users' sensitive data such as passwords when infected with attacked by malware. Their connectivity with mobile phones and the environment broadens their chances of being attacked by malware. These constitute security issues that need to be addressed.

9.1.3 Enhancing the security of smartphones through evidence analysis

Some smart malware automatically uninstall themselves and clean their logs after the attack goal is accomplished [7]. Analysis of left artefacts is integral for the detection and identification of potential and future transmission especially when the same attack vectors are used as identified in the LMD approach [8]. However, the use of anti-forensic techniques and tools on mobile and other smart devices to erase, alter, or conceal evidence makes this challenging and time-consuming for forensic and malware analysts. Although scheme such as replicas in the cloud [9] is in place as one of the data recovery approaches, a framework for evidence source neutralisation is needed to solve the problem of imperfect intentions [10] in the courtroom, workplace, among others.

Our results (**Sections 5.6.1, 5.7, 5.6.2**) demonstrated that malicious activities such as network connectivity, contacted URL can be extracted during analysis. This is significant in identifying threats (**Section 5.7**) posed by malicious applications. However, our research does not cover neutralisation approach of malware analysis. Neutralisation technique for digital forensic technique would help tackle the problem of blaming the limited choice, role, options, hiding behind

imperfect intentions, hiding behind imperfect capabilities and knowledge. For example, some criminals would intentionally infect a mobile device or corporate network with malware. When caught, they could leverage their imperfect capabilities or knowledge to handle malware or hide behind other factors such as circumstances to get away with their crime or reduce the sentence gravity. Although the Computer Misuse Act [11] spelled how such criminals should be given sentences. However, analysing the report of hackers who are convicted under the act shows that the Computer Misuse Act is limited to certain aspects for example when giving judgment to the unskilled group, juvenile offenders, mentally impaired, among others.

9.2 Summary of contributions

Malware is one of the major security challenges mobile and other security communities are faced with. If a digital device is infected by a malicious application, the application has control over the device's functionality and data. This can impact your finances, your privacy, and your access to your data. Malware is a threat not only to individuals but also to corporate organisations and financial bodies as well. Apart from financial impacts, malware attacks could lead to communication traffic of an infected network, hardware failure of the physical device, data theft, and loss of critical business data, among others. We carried out intensive research to identify the attack vectors and the infection strategies of evolving malware. we developed techniques to aid the prevention of malware infection on Android devices.

Our research findings generate potential impacts in academic, external, and social contexts respectively. In the context of academic impact, this research made a digital footprint as demonstrated by a record of requests from university institutions for the published research results (Appendix A). Findings of research make academic impacts if the results research other researchers or academic institutions as a recorded or audible occasion of influence. The research formulates methods and demonstrates they related to the existing research theories and applications on Android and mobile malware in general. In addition to building the academic knowledge base impact, the findings generate impacts outside the academic environment, which we referred to as the external impacts.

The published results are cited and referenced by security companies in their security reports on mobile malware threats and their economic impacts. This suggests research novelty and results from credibility and significance. This signifies that our research is gradually gaining an external reputation which indicates a track record of thorough and successful work. Finally, we believe that as the findings are being made public and cited by organizations and individuals, the outcome of this will influence the security and social policies that mobile companies will implement based on some of the recommendations by our findings. Major conclusions and outcomes resulting from the research contributions are summarised as follows:

9.2.1 Contribution 1: Paper 1

Among the cyber threats experienced during the COVID-19 pandemic, malware attacks especially on mobile devices were reported to be a global security challenge. We presented an analysis on mobile malware emergence, their evolution, and infection strategies (**Chapter 2, Paper 1 under review**) which provided the research background that formed the foundation for our thesis. We conducted an up-to-date study of the mobile malware evolution and infection strategies. Based on the exhibited characteristics, we classify mobile malware evolution trends into distinct categories based on their sophistication, characteristics, and attack intents. We provide a general overview of the mobile malware infection model using infection strategies meta-data. In (**Section 1.8.1**), factors limiting the fast spread of the early mobile malware: (a) the lack of mobile OS standardization, (b) lack of mobile OS cross-platform, and (c) lack of Bluetooth technological advancements in data communication were identified.

Our study identifies factors that enhance the fast spread of recent mobile malware. These include (a) the advancement in threat dimensions, (b) improved security and business communication on Tor using multi-signature transaction and encryption approach, (c) mobile platforms' integration with the IoT-based applications, (d) the emergence of evasion techniques and the advancements in mobile malware Toolkits on the dark web ((**Section 2.3.2.1**)). We provided a comprehensive survey on the history of malware (**Subsections 2.3.1**) by identifying major propagation modes and their present state. The results show that malware has evolved from mere fun to more sophisticated attacks such as cyberstalking, state-sponsored attacks, financial gain, among others. As discussed in (**Subsection 2.3.2**), the open-source nature of the Android operating system constitutes one of the major factors that facilitate malware infection and propagation mode.

We provided an extensive discussion on malware evasion techniques and their impacts on mobile application trust. Knowledge of the evasion techniques such as application repackaging is essential in understanding the vital starring role, they play in mobile malware proliferation.

Based on our contribution, our study provides recommendations as countermeasures to cope with the emerging mobile malware increasing threats (**Sections 2.7**). These include (a) the deployment of software solutions from enterprise mobility management to enhance the security of enterprise devices, (b) observation of mobile application plugins and codecs during mobile application download and installation, (c) the adoption of risk mitigation strategies by organizations, (d) development of a mobile malware infection model to understand the immunity state of a mobile device during and after recovery from malware infection. The comprehensive analysis suggested the need to analyse comprehensively and identify strengths and limitations in the existing detection techniques. The second research paper (Chapter 3) provides a broad but careful assessment of mobile malware detection and classification techniques. Strengths and limitations of the detection techniques were identified and carefully analysed to help develop a more robust tool for the identification of sophisticated mobile malware variants.

9.2.2 Contribution 2: Paper 2

As malware is becoming smarter and sophisticated, there is a need to improve on existing techniques to enhance detection and classification. Part of the literature review (**Chapter 3**) published (**Paper 2**) provided a comprehensive review of the existing Android malware detection detections. The paper provided background on mobile malware with a comparative analysis of the static, dynamic, and machine learning approaches (**Section 3.2**) for detecting mobile malware. The detection techniques (**Section 3.4**) examined were found to have unique strengths and limitations which are highlighted in **Table 3-1**. One of the major common limitations among the techniques is their susceptibility to transformation and zero-day attacks as highlighted in the evaluation (**section 3.6**). One of the things we did not cover in the review is a comparison of the techniques with those used by other mobile platforms such as iOS. During the comparison, parameters optimization was not considered as a criterion which we suggested for further research.

Identification of strengths and limitations in the existing techniques is significant to building an efficient and robust detection framework by leveraging the identified strengths. Also, the identified weaknesses are as well significant for future research for improvement for better detection. In summary, this chapter (**Paper 2**) serves as a good starting point for research on the limitations and strengths of detection tools apart from identifying significant research gaps. Findings of research make academic impacts if the results reach other researchers or academic institutions as a recorded or audible occasion of influence. This chapter impacts academic advances across and within the mobile discipline. Also, the published results are cited and referenced by security companies in their security reports. This demonstrates the research's credibility and significance as it gains an external reputation and track record of impactful work.

9.2.3 contribution 3: Building malware lab

Analysis of malware in an uncontrolled environment is very disastrous as it could cause escalation to the outside world. Smart and sophisticated variants such as polymorphic malware change their functionalities such as file name or attributes in certain instances to circumvent detection by signature approach. As a result, there is a need for a risk management plan during malware analysis to avoid critical infrastructure from disruption and destruction by malware. To avoid regulatory, reputational, and legal risk, we built a malware lab (**Chapter 4**) using sandboxing technique and change control for monitoring both the host and guest OS. Security configurations ranging from network configurations and privileged accounts set up (**Section 4.2**) were established. Tools and rules were configured to monitor both inbound and outbound traffic. This helps to monitor for escalation during analysis.

The malware lab built for this research forms a starting point where other researchers whose research interest is on malware analysis could be leveraged on. By integrating both virtual and physical devices, the lab aids evaluation and automated analysis of sophisticated and new malware variants and their features. The major strength of our lab is that though the design is simple and cost-effective, and the lab shows a high degree of containment. No issue of malware escalation was reported throughout the research period. In addition, the lab is flexible and easy to maintain. Although the maintenance of the lab may not be a guarantee, the containment strength is more vital. Our lab, however, was not tested with malware data set and as a result, does not guarantee containment of other malware datasets apart from Android. In addition, security patches were not

applied during the development and as a result, security vulnerabilities need to be tested been used as a commercial product.

9.2.4 Contribution 4: Paper 3

The third research paper (Chapter 5) presents how malware can be classified by statistically analysing malware with Linux malware detect (LMD) technique. LMD is a technique for mining text features by configuring some functions with rules. The technique retrieves data and statistically analyse the code if obfuscated using encoding methods. We accustom the ClamAV approach in the reformation of the model process for similarity classification. LMD technique extracted detected privileges exploited by sophisticated malware families. We identified permission requests such as HTTP requests, text messages, and network connection during synchronization, payloads executed, location monitoring, and battery consumption. Understanding how malware escalates privileges is significant in combating malware because it will shift the prevention paradigm from proactive to reactive. This contribution is significant because it discloses how privileges are exploited by emerging smart malware families especially from the developmental application reuse point of view. In addition, this contribution facilitates the measurement of malware file similarities which can be used to classify malware into their respective families automatically. It also aids malicious code module identification with their similarities during an investigation by malware experts.

9.2.5 Contribution 4: Paper 4

Permission is one of the major security architectures Android uses to prevent different forms of threats. Mobile applications request different permissions which may be normal or dangerous. Automated classification of malicious applications using permission attributes is essential in combating malware. Existing techniques do not give a good classification accuracy, especially when dealing with sophisticated malware such as polymorphic variants. As a result, we proposed a classification framework (**Chapter 6**) published in Paper 4 to optimise Android malware classification based on permission attributes. The framework has the capacity to use other attributes such as intent and command signature to train and learn in predicting the new trend of smart malware.

We improved the classification accuracy of our framework by selecting only features that are most important using feature classification (**Section 6.5**). One of the major strengths of our framework is feature diversity and threshold capabilities (**Sections 6.7.2**). The feature diversity method was used to reduce feature redundancy and minimise classification error in the framework. The diversity threshold ensured adequate differentiation of feature vectors in the data set. The framework demonstrated results with high accuracy with significantly low feature space when compared to the state-of-the-art methods.

Identifying dangerous permissions frequently requested by malicious applications plays an essential role in developing and building a secure mobile computing ecosystem. It enhances malware detection and data protection. Permission request depicts application's characteristic patterns, especially in Android platform. This contribution provides malware analysts and digital forensic investigators with an efficient framework for automated analysis of malicious. For further research, we suggest testing the model using data set from other mobile platforms.

9.2.6 Contribution 4: Paper 5

Dangerous permissions that malicious applications request to access the device resources of a victim exhibits similar attributes. Even though permission is granted, an application may not have access to a device resource based on the Android permission architecture. However, permissions with similar attributes could trigger background actions because of similarities in their clusters. As a result, there is a need to investigate the relationship between permissions to identify their threat level. This contribution is provided in **Paper 5, Chapter 7**. We proposed a model that demonstrated how risky some malicious applications are based on their threat level (**Section 7.4**) using exploratory factor plane analysis (**Section 7.3.4**). The model was implemented using t-SNE (**Subsection 7.3.2**) and SOM (**Subsection 7.3.3**) algorithms respectively.

Visualisation results obtained from the model (**Section 7.3.4**) indicate that most permissions in the large subcategory were requested by very few Android applications while frequently used permissions were those in the small subcategory. We proposed a protection level and permission-based taxonomy with flags that indicate the presence of dangerous permission during analysis. We identified that there are permissions with similar attributes which can trigger the installation of similar permissions with the same threat level in the background without activating further request actions for access and payloads execution.

The chapter identifies that in a random sample of a given malicious dataset, some features of some variables have control or influence the characteristics of one or more variables at given levels of independent measurement. The results of the contributors demonstrate that there is no adequate expressiveness from permissions that are requested frequently. The importance of this contribution will assist in enforcing security parameters at the permission security system of the Android platform and other mobile OS. This understanding will aid mobile users to prioritize their privacy over the urge to use the application especially when the application is not from trusted markets or sites. These findings are significant for offering insights to assist mobile users to make low-risk decisions during application installations and when granting access to applications that have high threat levels (**Sections 7.5.1**).

Our model can be expanded to other mobile permission-based platforms such as Windows and Symbian. We infer those permissions not frequently requested could be disintegrated into the common class. As a result, we suggest further research on adding finer granularity as a security approach for Android permissions that are frequently requested by applications. I hope adding finer granularity will enhance this expressiveness and enhance Android security, especially when combined with the ones occasionally requested. However, we are not sure if finer granularity will increase the model complexity especially when implemented using different data.

9.3 Conclusion

As of August 2021, Android remained the most popular and leading OS in the mobile OS platform market with an increasing number of threats such as malware that targets the OS. While factors like cheap to purchase and ease of operation enhance its popularity, the open-source nature of Android enhances malware infection. Android open-source architecture allows developers to build application variants that meet the compatibility of the Android requirements. While the goal is to contribute code that will enhance the functionality and security of the project, hackers repackage malicious codes that can not be easily reviewed. Consequently, among other factors, this has made malware on Android keep growing rapidly. Thus, it is critical to investigate malware evolution and infection strategies by evaluating the existing classification, detection, and analysis methods to improve detection and if possible, prevent malware attacks on Android.

The author introduced the thesis by presenting recent cases of major malware attacks and comprehensive reviews on Android security, malware state of the art, and detection techniques. The review identified the weaknesses and strengths in the existing techniques that require improvement for detecting smart and sophisticated malware. This lead to building a framework that classifies Android permissions. To appraise our research value and areas of our research impacts, the weaknesses and strengths of our techniques are compared with traditional frameworks. Our techniques showed high diversity in the malicious feature classification. The framework robustly identified some attack vectors which are privileges that are escalated by smart malware to infect Android. Classification accuracy of 97% was achieved with our framework.

To reflect on the work, the framework was tested by using methods that could classify malicious applications based on their permissions and also identify the correlation between permission variables. We performed an exploratory analysis to determine the threat and protection level relationship in each permission. This research applied permission forensic analysis in a different approach of detecting and classifying malware on the Android platform. The results of our methods showed clearly their robust and scalable capabilities for detecting and mitigating smart malware that use permissions to escalate privileges. We conclude our thesis that the research question “*Is it possible to develop a framework that identifies attack vectors and privileges on the permission request system that smart malware deploys to infect Android-based platforms, to prevent malware attacks on Android OS devices?*” has been answered.

REFERENCES

- [1] W. J. Buchanan, S. Chiale, and R. Macfarlane, "A methodology for the security evaluation within third-party Android Marketplaces," *Digit. Investig.*, vol. 23, pp. 88–98, 2017, doi: 10.1016/j.diin.2017.10.002.
- [2] M. Ashawa, S. Morris, "Android Permission Classifier: a deep learning algorithmic framework based on protection and threat levels," *Int. Jour. of Sec. and Priv.*, vol. 4, no. 5, pp. 1–26, 2021, doi: 10.1002/spy2.164.
- [3] K. Govindan and P. Mohapatra, "Trust computations and trust dynamics in mobile adhoc networks: A survey," *IEEE Commun. Surv. Tutorials*, vol. 14, no. 2, pp. 279–298, 2012, doi: 10.1109/SURV.2011.042711.00083.
- [4] R. Tobias, A. Höller, N. Kajtazovic, and C. Kreiner. "Privilege-based remote attestation: Towards integrity assurance for lightweight clients." In *Proceedings of the 1st ACM Workshop on IoT Privacy, Trust, and Security*, pp. 3-9. 2015, doi: 10.1145/2732209.2732211.
- [5] A. Baranchuk et al., "Cybersecurity for Cardiac Implantable Electronic Devices: What Should You Know?," *J. Am. Coll. Cardiol.*, vol. 71, no. 11, pp. 1284–1288, 2018, doi: 10.1016/j.jacc.2018.01.023.
- [6] W. Burleson and S. Carrara, "Security and privacy for implantable medical devices," *Secur. Priv. Implant. Med. Devices*, vol. 97, pp. 1–205, 2013, doi: 10.1007/978-1-4614-1674-6.
- [7] G. Suarez-Tangil, J. E. Tapiador, P. Peris-Lopez, and A. Ribagorda, "Evolution, detection and analysis of malware for smart devices," *IEEE Commun. Surv. Tutorials*, vol. 16, no. 2, pp. 961–987, 2014, doi: 10.1109/SURV.2013.101613.00077.
- [8] M. Ashawa and S. Morris, "Host-Based Detection and Analysis of Android Malware: Implication for Privilege Exploitation," *Int. J. Inf. Secur. Res.*, vol. 9, no. 2, pp. 871–880, 2019, doi: 10.20533/ijisr.2042.4639.2019.0100.
- [9] S. Patil, R. Dharaskar, and V. Thakare, "Digital Forensics Technique for Detection of Attack and Previous Data Restoration in Cloud Environment," *Int. J. Adv. Res. Comput. Sci. Softw. Eng.*, vol. 7, no. 6, pp. 427–433, 2017, doi: 10.23956/ijarcsse/v7i6/0125.
- [10] M. Kaptein and M. van Helvoort, "A Model of Neutralization Techniques," *Deviant Behav.*, vol. 40, no. 10, pp. 1260–1285, 2019, doi: 10.1080/01639625.2018.1491696.
- [11] Criminal Law Reform Now Network, "Reforming the Computer Misuse Act 1990," *Crim. Law Reform Now Netw. Rep.*, pp. 1–161, 2020, Accessed on 17 July, 2021. [Online]. Available: <http://www.clrnn.co.uk/media/1018/clrnn-cma-report.pdf>.

Appendix A: CERES request for a copy service

CERES Request a copy service : Analysis of Android malware detection techniques: a systematic review



Hinton, Sharon

Wed 03/06/2020 9:56 AM

To: Ashawa, Moses Aprofim; Morris, Sarah



Hello Moses and Sarah,

RE: Analysis of Android malware detection techniques: a systematic review

<http://dspace.lib.cranfield.ac.uk/handle/1826/15057>

<http://sdiwc.net/digital-library/analysis-of-android-malware-detection-techniques-a-systematic-review>

I have received a request for your article through the CERES request-a-copy service

The requestor message reads:

FernUniversität in Hagen [...] aiming for a masters degree in computer science. I already have a bachelors degree.

We are able to supply the author accepted manuscript (AAM) that is held in CRIS for personal use only.

However, as the AAM of the article is still under the publisher embargo (for Green OA), I just need to know that you are happy for us to share this version of the article.

Please get back to me asap so I can reply to the requestor.

Many thanks,

Sharon

Coronavirus: Please visit our page to find out more about [how our services are changing as we move to an online service](#).

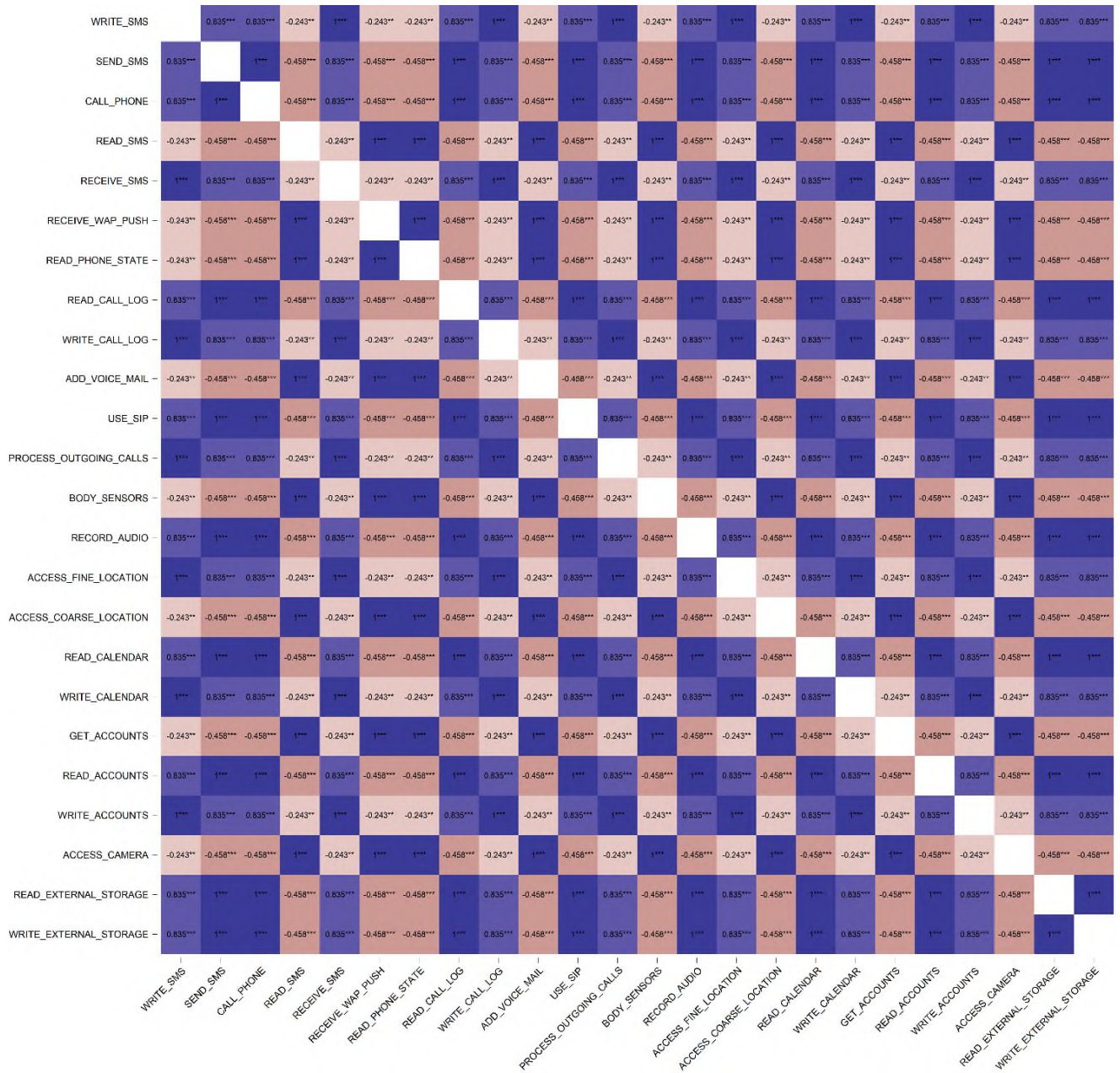
Sharon Hinton Gallardo

Senior Information Advisor - Open Access

Kings Norton Library, Cranfield University, Cranfield, Bedfordshire, MK43 0AL

W: www.cranfield.ac.uk | E: S.Hinton@cranfield.ac.uk

Appendix B: Correlation heatmaps



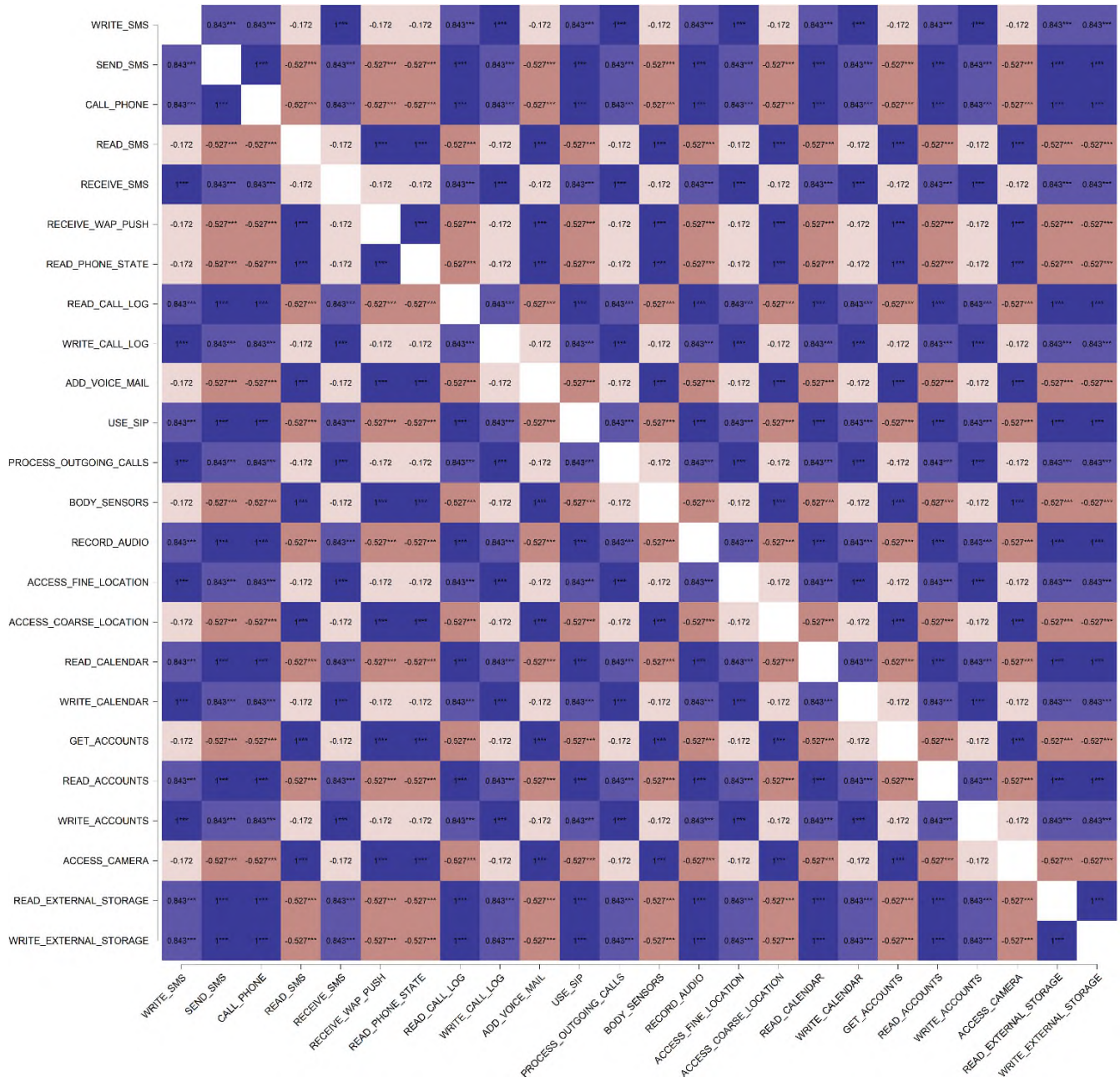


Figure B-2: Pearson's r correlation heatmap

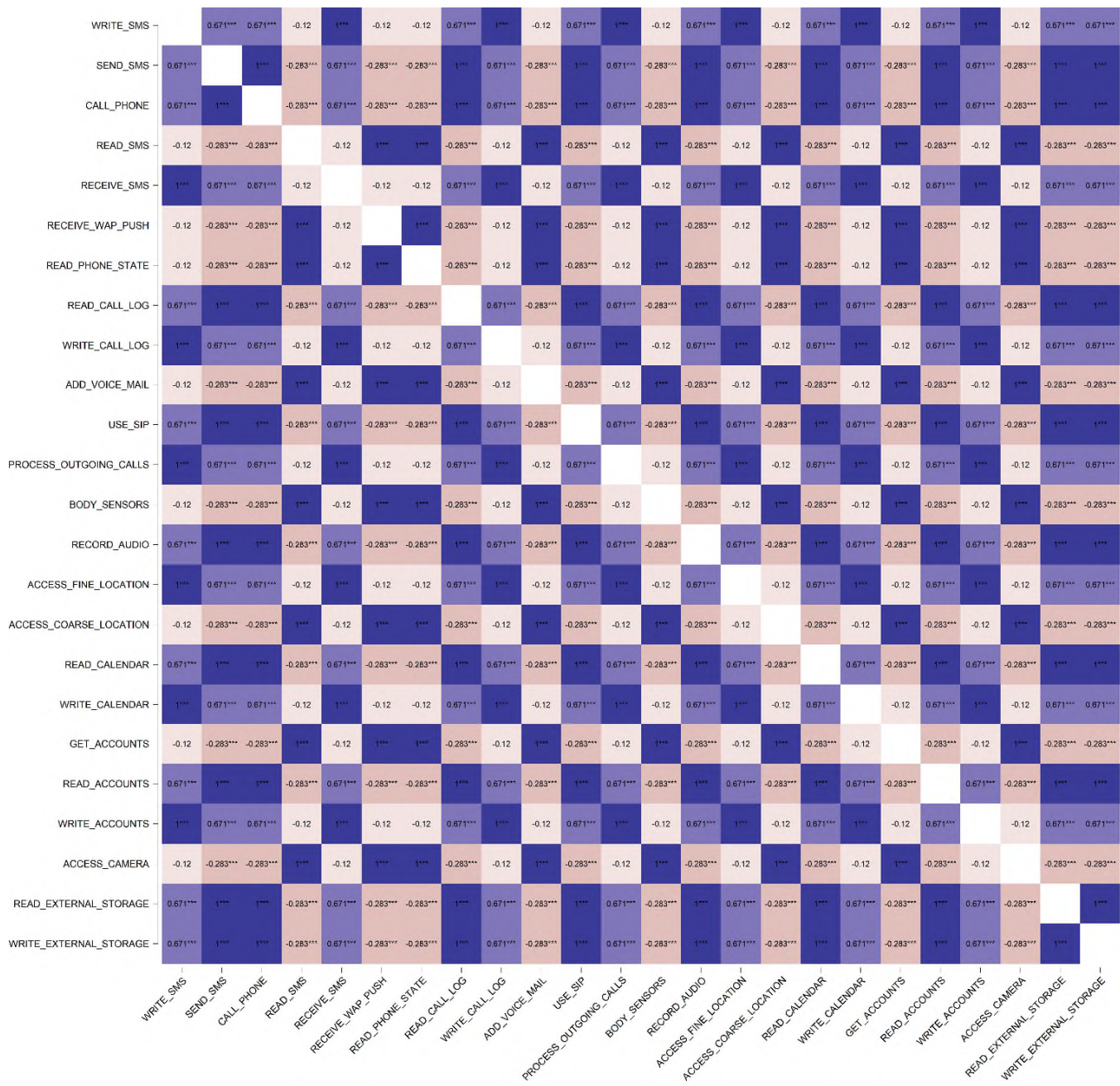


Figure B-3: Kendall's tau-b correlation heatmap